Tessent® Shell Reference Manual

Software Version 2014.2

June 2014

This manual is part of a fully-indexed Tessent documentation set. To search across all Tessent manuals, click on the “binocular” icon or press Shift-Ctrl-F. Note that this index is not available if you are viewing this PDF in a web browser.

©2011-2014 Mentor Graphics Corporation
All rights reserved.

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

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

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

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

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

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

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

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

Mentor Graphics Corporation
8005 S.W. Boeckman Road, Wilsonville, Oregon 97070-7777
Telephone: 503.685.7000
Toll-Free Telephone: 800.592.2210
Website: www.mentor.com
SupportNet: supportnet.mentor.com/

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

Table of Contents

Chapter 1
Introduction 39

Overview 41
Tessent FastScan and Tessent TestKompress 41
Tessent DesignEditor 43
Tessent Scan 44
Tessent Diagnosis 45
Tessent IJTAG 45
Optimization of Adobe Reader 46

Chapter 2
Shell Commands 47

Shell Command Descriptions 47
create_skeleton_design 48
stil2mgc 50
tessent 52

Chapter 3
Data Models 57

Introduction 57
Hierarchical Design Data Model 58
Hierarchical Object Types 58
Module 59
Instance 64
Port 67
Pin 72
Net 75
Pseudo_port 78
Flat Design Data Model 80
Flat Design Object Types 80
Gate_pin 81
ICL Data Model 87
ICL Object Types 87
icl_module 88
icl_port 93
icl_instance 98
icl_pin 100

Chapter 4
Attributes 103

Attribute-Related Commands 103
Attribute Types 104

Pre-Defined Attributes 104
User-Defined Attributes 104
Attribute Value Types 104
Attribute Filtering Equation Syntax 105
Glob and Regular Expression Pattern Matching Syntax 107
Example 1 109
Example 2 110
Example 3 110
Example 4 110
Matching Hierarchical Names With Escaped Identifiers 111

Chapter 5
Configuration-Based Specification 113

DftSpecification Configuration Syntax 113
DftSpecification 114
IjtagNetwork 116
DesignInstance 120
HostScanInterface/Interface 123
IjtagNetwork/DataInPorts 129
IjtagNetwork/DataOutPorts 132
ScanMux 134
ScanMux/Interface 140
Sib 143
Sib/Interface 149
Tap 152
Tap/Interface 157
Tap/DataOutPorts 160
Tap/DeviceIDRegister 163
Tap/HostBScan 165
Tap/HostIjtag 169
Tdr 171
Tdr/DataInPorts 177
Tdr/DataOutPorts 179
Tdr/DecodedSignal 182
Tdr/Interface 185
DefaultsSpecifications Configuration Syntax 187
DefaultsSpecification/DftSpecification 188
IjtagNetwork 189
Metadata Configuration Syntax 193
Wrapper 194
Property 201
RepeatableProperty 205
DataWrapper 208
CsvWrapper 211
Id 213
Value 216
RowIdElement 219

Chapter 6
Instrument Connectivity Language (ICL) 221

Instrument Connectivity Language Syntax 221
Numbers 221
Identifiers 222
Limited Identifier Mathematics 223
Concatenations 223
Syntax Conventions 224
Module 225
Attribute 227
ScanInterface. 228
Chain. 233
ScanInPort. 236
ShiftEnPort 238
CaptureEnPort. 240
UpdateEnPort 242
DataInPort. 244
SelectPort 247
ResetPort. 249
TMSPort 251
TCKPort 254
ClockPort 255
TRSTPort 257
AddressPort. 258
WriteEnPort 259
ReadEnPort 260
ScanOutPort 262
DataOutPort 264
ToShiftEnPort 267
ToCaptureEnPort 270
ToUpdateEnPort 273
ToResetPort. 276
ToSelectPort 279
ToClockPort 282
ToTCKPort 286
ToTMSPort 288
ToTRSTPort 290
ToIRSelectPort 292
Instance 293
Parameter 298
LocalParameter 301
Enum. 302
Alias 305
LogicSignal 308
ScanRegister 310
DataRegister 314
ScanMux 319
DataMux 322

ClockMux 324
OneHotDataGroup 326
OneHotScanGroup 329
AccessLink 331
NameSpace 339
UseNameSpace 340

Chapter 7
Command Dictionary 341

Command Line Syntax Conventions 341
Command Summary 342
Command Descriptions 382
add_ambiguous_paths 384
add_atpg_constraints 386
add_atpg_functions 390
add_bist_capture_range 393
add_black_boxes 394
add_browser_data 398
add_capture_handling 401
add_cdp_test 404
add_cell_constraints 405
add_cell_models 412
add_chain_masks 415
add_clocks 417
add_config_element 424
add_config_message 429
add_config_tab 430
add_control_points 431
add_core_instances 433
add_display_data 438
add_display_instances 440
add_edt_blocks 445
add_edt_configurations 448
add_edt_connections 450
add_false_paths 453
add_fault_sites 459
add_faults 461
add_icl_ports 470
add_icl_scan_interfaces 472
add_iddq_exceptions 474
add_ijtag_logical_connection 476
add_input_constraints 480
add_lfsr_connections 485
add_lfsr_taps 487
add_lfsrs 489
add_lists 492
add_misrs 494
add_nofaults 495

add_nonscan_instances 498
add_nonscan_models 500
add_norace_clocks 501
add_notest_points 503
add_observe_points. 505
add_output_masks. 507
add_primary_inputs 509
add_primary_outputs 512
add_processors 514
add_read_controls 517
add_register_value 518
add_scan_chains 520
add_scan_groups. 523
add_scan_instances 525
add_scan_models 527
add_scan_partition 529
add_scan_pins 534
add_simdut_fault. 538
add_simulation_context 540
add_simulation_force 541
add_sub_chains 543
add_subchain_clocks 548
add_subchain_group 553
add_tied_signals 556
add_to_collection 558
add_write_controls 560
analyze_atpg_constraints 561
analyze_bus. 564
analyze_compression 568
analyze_control_signals 583
analyze_drc_violation 586
analyze_fault 588
analyze_graybox 594
analyze_restrictions. 597
analyze_simulation_mismatches. 600
analyze_test_points 608
analyze_wrapper_cells 610
analyze_xbounding 612
annotate_diagnosis 614
append_to_collection 615
apply_specification_defaults. 617
catch_output 619
close_layout 621
close_pattern_set. 622
close_tsdb 624
close_visualizer. 625
compare_collections 626
compress_layout 628
compress_patterns. 629

copy_collection 632
copy_module 634
copy_simulation_context 636
create_capture_procedures 637
create_connections 641
create_dft_specification 644
create_diagnosis_patterns 652
create_feature_statistics 654
create_flat_model 656
create_initialization_patterns 658
create_instance 660
create_layout 663
create_module 667
create_net 669
create_patterns 671
create_pin 676
create_port 678
delete_atpg_constraints 680
delete_atpg_functions 682
delete_bist_capture_ranges 683
delete_black_boxes 684
delete_browser_data 686
delete_capture_handling 688
delete_capture_procedures 690
delete_cdp_test 691
delete_cell_constraints 692
delete_cell_library 694
delete_cell_models 695
delete_chain_masks 697
delete_clocks 698
delete_config_elements 699
delete_config_messages 700
delete_config_tabs 702
delete_connections 703
delete_core_descriptions 705
delete_core_instances 707
delete_design 709
delete_dfm 710
delete_display_data 711
delete_display_instances 713
delete_edt_blocks 715
delete_edt_configurations 718
delete_edt_connections 720
delete_false_paths 722
delete_fault_sites 725
delete_faults 729
delete_flat_model 738
delete_icl_modules 739
delete_icl_ports 740

delete_icl_scan_interfaces 741
delete_iddq_exceptions 742
delete_ijtag_logical_connection 743
delete_input_constraints 745
delete_instances 748
delete_iproc 749
delete_layout_verification 750
delete_lfsr_connections 752
delete_lfsr_taps 753
delete_lfsrs 754
delete_lists 755
delete_misrs 756
delete_multicycle_paths 757
delete_nets 760
delete_nofaults 762
delete_nonscan_instances 765
delete_nonscan_models 767
delete_norace_clocks 769
delete_notest_points 771
delete_output_masks 773
delete_patterns 774
delete_pins 776
delete_ports 778
delete_primary_inputs 780
delete_primary_outputs 782
delete_processors 784
delete_read_controls 786
delete_register_value 787
delete_scan_chains 788
delete_scan_groups 789
delete_scan_instances 790
delete_scan_models 792
delete_scan_partitions 793
delete_scan_pins 795
delete_simdut_fault 796
delete_simulation_contexts 797
delete_simulation_forces 799
delete_sub_chains 800
delete_subchain_clocks 801
delete_subchain_groups 802
delete_test_points 803
delete_tied_signals 804
delete_write_controls 806
diagnose_failures 807
display_specification 810
display_diagnosis_report 812
display_message 813
dofile 815
echo 817

execute_cdp_test 818
execute_gpib_command 820
exit. 822
expand_compressed_patterns 823
extract_icl 824
filter_collection 826
find_design_names 828
foreach_in_collection 833
get_attribute_list 836
get_attribute_option 838
get_attribute_value_list 840
get_cdp_test_list 842
get_clock_list 843
get_clock_option 844
get_common_parent_instance. 846
get_config_elements 847
get_config_messages 851
get_config_selected_nodes 853
get_config_value 855
get_context 867
get_current_design 870
get_current_mode 871
get_current_simulation_context 872
get_defaults_value 873
get_design_level 875
get_design_sources 876
get_dfm_rules 878
get_dft_cell 880
get_fanins 889
get_fanouts 892
get_fault_type 895
get_gate_pins 896
get_icl_extraction_options 900
get_icl_fanins 901
get_icl_fanouts 903
get_icl_instances 905
get_icl_modules 909
get_icl_module_parameter_list 911
get_icl_module_parameter_value 912
get_icl_pins 913
get_icl_ports 917
get_icl_scan_interface_list 919
get_icl_scan_interface_port_list 921
get_icl_scope. 923
get_iclock_list 925
get_iclock_option 927
get_ijtag_instances 929
get_ijtag_instance_option 931
get_ijtag_retargeting_options 932

get_insertion_option 934
get_insert_test_logic_options 936
get_instances 937
get_iproc_argument_default 941
get_iproc_argument_list 943
get_iproc_body 944
get_iproc_list 945
get_license_queue_timeout 946
get_logical_library_list 947
get_module_parameter_list 948
get_module_parameter_value 949
get_modules 950
get_multiprocessing_option 954
get_name_list 955
get_nets 957
get_open_pattern_set 960
get_pattern_cycle_count 961
get_pattern_set_data 962
get_pattern_set_list 969
get_pattern_set_option 970
get_pins 972
get_ports 975
get_procfile_name 979
get_resource 980
get_scratch_directory 982
get_simulation_context_list 983
get_simulation_value_list 984
get_single_name 985
get_system_mode 987
get_test_end_icall_list 988
get_test_setup_icall_list 989
get_tool_info 990
get_trace_flat_model_option 992
get_transcript_style 994
get_tsdb_list 995
get_tsdb_output_directory 996
help 997
history 999
iApply 1001
iCall 1004
iClock 1007
iClockOverride 1009
identify_redundant_faults 1012
iMerge 1015
import_dfm 1017
index_collection 1019
iNote 1020
insert_test_logic 1021
intercept_connection 1027

iPrefix 1036
iProc 1038
iProcsForModule. 1040
iRead 1042
iRelease 1044
iReset. 1045
iRunLoop 1047
is_collection 1050
iTake 1052
iUseProcNameSpace. 1053
iWrite 1054
launch_sid_tester. 1056
lock_current_registration 1058
macrotest 1059
mark_display_instances 1066
migrate_layout 1068
move_config_element. 1069
move_connections. 1073
no_transcript 1079
open_layout. 1082
open_pattern_set 1085
open_tsdb 1092
open_visualizer 1094
order_patterns 1096
printenv 1099
process_dft_specification 1100
range_collection 1104
read_cell_library 1105
read_config_data. 1106
read_core_descriptions 1109
read_cpf. 1110
read_failures 1112
read_fault_sites 1120
read_faults. 1130
read_flat_model 1139
read_icl 1140
read_modelfile 1142
read_patterns 1146
read_procfile 1151
read_sdc 1156
read_sdf. 1163
read_upf 1165
read_verilog 1166
read_vhdl. 1170
read_visualizer_preferences 1173
read_window_contents 1174
register_attribute 1176
register_tcl_command. 1179
remove_from_collection. 1189

rename_instance 1190
replace_instances 1192
report_aborted_faults 1194
report_atpg_constraints 1196
report_atpg_functions 1197
report_atpg_timing 1198
report_attributes 1201
report_bist_capture_ranges 1207
report_black_boxes 1208
report_bus_data 1210
report_bypass_chains 1213
report_capture_handling 1216
report_capture_procedures 1218
report_cell_constraints 1224
report_cell_models 1227
report_chain_masks 1229
report_clock_controller_pins 1230
report_clock_controls 1231
report_clock_domains 1232
report_clock_gating 1237
report_clocks 1239
report_compactor_connections 1243
report_config_data 1246
report_config_messages 1249
report_config_syntax 1251
report_context 1254
report_control_signals 1256
report_core_descriptions 1258
report_core_instances 1259
report_design_sources 1262
report_dfm_rules 1263
report_dft_check 1265
report_diagnosis 1268
report_display_instances 1269
report_drc_rules 1272
report_edt_blocks 1279
report_edt_configurations 1280
report_edt_connections 1285
report_edt_finder 1287
report_edt_instances 1292
report_edt_lockup_cells 1295
report_edt_pins 1300
report_environment 1303
report_external_simulator 1305
report_failures 1306
report_false_paths 1310
report_fault_sites 1315
report_faults 1323
report_feedback_paths 1336

report_flattener_rules 1338
report_gates. 1340
report_graybox_statistics 1361
report_icl_extraction_options 1363
report_icl_modules 1364
report_iclock 1368
report_id_stamp 1369
report_iddq_exceptions. 1371
report_ijtag_instances 1372
report_ijtag_logical_connections 1374
report_ijtag_retargeting_options. 1376
report_input_constraints 1377
report_layout_rules 1380
report_lbist_configuration. 1384
report_lbist_pins 1385
report_lfsr_connections. 1386
report_lfsrs 1387
report_licenses 1388
report_lists. 1389
report_loops 1390
report_lpct_configuration 1392
report_lpct_pins 1393
report_measure_cycles 1394
report_mismatch_sources 1399
report_misrs 1403
report_misr_connections. 1405
report_module_matching 1407
report_module_matching_options 1409
report_multicycle_paths 1410
report_multiprocessing_options 1415
report_nofaults 1417
report_nonscan_cells. 1419
report_nonscan_models 1421
report_norace_clocks 1422
report_notest_points 1423
report_output_masks. 1424
report_pattern_filtering. 1425
report_pattern_sets 1426
report_patterns 1427
report_power_data 1435
report_power_metrics 1437
report_primary_inputs. 1443
report_primary_outputs. 1445
report_procedures 1447
report_processors 1449
report_read_controls 1451
report_register_value 1452
report_resources 1455
report_scan_cells. 1457

report_scan_chains 1465
report_scan_enable 1469
report_scan_groups 1470
report_scan_models 1471
report_scan_partitions 1472
report_scan_pins 1475
report_scan_volume 1476
report_seq_transparent_procedures 1479
report_sequential_fault_depth 1481
report_sequential_instances 1483
report_shift_registers 1489
report_simulation_contexts 1491
report_simulation_forces 1493
report_statistics 1494
report_sub_chains 1506
report_subchain_clocks 1507
report_subchain_groups 1508
report_test_logic 1509
report_test_points 1511
report_test_stimulus 1513
report_testability_data 1522
report_tied_signals 1525
report_timeplate 1526
report_udfm_statistics 1527
report_version_data 1531
report_wrapper_cells 1532
report_write_controls 1537
report_xbounding 1538
reset_attribute_value 1540
reset_au_faults 1541
reset_bypass_chains 1543
reset_compactor_connections 1544
reset_design 1545
reset_di_faults 1546
reset_open_pattern_set 1549
reset_state 1550
restore_design 1551
return_output 1552
select_display_instances 1553
set_abort_limit 1555
set_atpg_compression 1557
set_atpg_fill 1560
set_atpg_limits 1561
set_atpg_timing 1564
set_attribute_options 1567
set_attribute_value 1571
set_au_analysis 1573
set_bidi_gating 1574
set_bist_chain_test 1577

set_bist_debug. 1578
set_bist_trace. 1582
set_bus_handling. 1585
set_bus_simulation 1588
set_bypass_chains 1590
set_capture_clock 1593
set_capture_handling 1596
set_capture_procedures. 1598
set_cell_library_options 1600
set_cell_model_mapping 1601
set_chain_test 1603
set_checkpointing_options 1606
set_clock_controller_pins 1608
set_clock_controls. 1611
set_clock_gating 1614
set_clock_gating_enable 1617
set_clock_off_simulation 1619
set_clock_restriction 1620
set_compactor_connections 1626
set_config_value 1628
set_contention_check 1631
set_context 1636
set_current_design 1640
set_current_edt_block 1642
set_current_edt_configuration 1644
set_current_mode 1646
set_current_simulation_context 1648
set_decision_order 1649
set_defaults_value. 1650
set_design_include_directories 1651
set_design_level 1652
set_design_macros 1653
set_design_sources 1654
set_dft_enable_options 1657
set_diagnosis_options 1658
set_display. 1668
set_dofile_abort. 1670
set_drc_handling 1672
set_driver_restriction 1679
set_edt_finder 1681
set_edt_instances. 1683
set_edt_mapping 1686
set_edt_options 1689
set_edt_pins. 1697
set_edt_power_controller 1707
set_external_capture_options 1710
set_external_simulator 1713
set_fails_report 1715
set_fault_mode 1716

set_fault_protection. 1718
set_fault_sampling 1720
set_fault_subclass_analysis. 1722
set_fault_type 1725
set_file_compression. 1732
set_flattener_rule_handling. 1734
set_gate_level 1737
set_gate_report 1738
set_gzip_options 1767
set_icl_extraction_options 1769
set_icl_scan_interface_ports 1770
set_iddq_checks 1772
set_iddq_strobe 1776
set_ijtag_instance_options 1777
set_ijtag_retargeting_options 1778
set_insertion_options 1780
set_insert_test_logic_options 1783
set_internal_fault. 1784
set_internal_name 1785
set_io_mask. 1786
set_latch_handling 1787
set_lbist_controller_options 1788
set_lbist_instances. 1791
set_lbist_pins 1792
set_lbist_power_controller_options 1796
set_learn_report. 1798
set_lfsr_seed 1800
set_lfsrs 1801
set_license_queue_timeout 1803
set_list_file 1804
set_lockup_cell 1805
set_logfile_handling 1810
set_logical_design_libraries 1812
set_loop_handling 1814
set_lpct_condition_bits 1816
set_lpct_controller. 1818
set_lpct_instances 1821
set_lpct_pins 1822
set_macrotest_options. 1830
set_misr_connections 1832
set_module_matching_options 1834
set_multiple_detection 1836
set_multiprocessing_options. 1840
set_name_prefix 1846
set_net_dominance 1849
set_net_resolution 1851
set_number_shifts 1852
set_observation_point 1854
set_output_masks 1855

set_parallel_load_subchains 1859
set_pathdelay_holdpi 1861
set_pattern_buffer 1862
set_pattern_classification 1863
set_pattern_filtering 1864
set_pattern_source. 1873
set_pattern_type 1883
set_physical_translation 1889
set_possible_credit 1891
set_power_control. 1892
set_power_domain 1896
set_power_metrics 1897
set_procedure_cycle_checking 1901
set_procedure_retargeting_options 1902
set_procfile_name 1904
set_ram_initialization 1906
set_ram_test 1907
set_random_atpg 1908
set_random_clocks 1909
set_random_patterns 1911
set_register_value 1912
set_registered_io 1913
set_relevant_coverage. 1920
set_scan_enable. 1924
set_scan_enable_sharing. 1929
set_scan_insertion 1934
set_scan_pins 1937
set_screen_display 1940
set_shadow_check. 1941
set_shift_register_identification 1942
set_simulation_options 1943
set_skewed_load 1946
set_split_capture_cycle 1947
set_stability_check 1948
set_static_learning. 1951
set_system_mode 1953
set_test_end_icall 1955
set_test_logic. 1957
set_test_point_analysis_options 1959
set_test_point_insertion_options 1963
set_test_setup_icall 1965
set_tied_signals 1967
set_timing_exceptions_handling. 1968
set_tla_loop_handling 1971
set_trace_flat_model_options 1972
set_trace_report. 1975
set_transcript_style 1976
set_transient_detection 1979
set_transition_holdpi. 1981

set_tristate_gating 1982
set_tsdb_output_directory 1985
set_visualizer_logging 1986
set_visualizer_preferences 1987
set_wrapper_chains 1989
set_xbounding_options 1997
set_xclock_handling 2000
set_z_handling 2002
set_zhold_behavior 2004
setenv 2005
shutdown_sid_tester 2006
simulate_clock_pulses 2007
simulate_forces 2008
simulate_patterns 2009
sizeof_collection 2011
sort_collection 2013
system 2015
trace_flat_model 2016
uncompress_layout 2023
uniquify_instances 2024
unmark_display_instances 2026
unregister_attribute 2028
unregister_tcl_command 2029
unselect_display_instances 2030
unsetenv 2031
update_implication_detections 2032
verify_patterns 2033
write_atpg_setup 2037
write_cell_library 2040
write_config_data 2042
write_core_description 2045
write_core_timing_constraints 2047
write_design 2048
write_diagnosis 2053
write_edt_files 2056
write_failing_paths 2067
write_failures 2070
write_fault_sites 2074
write_faults 2078
write_flat_model 2087
write_icl 2090
write_loops 2092
write_modelfile 2093
write_patterns 2094
write_primary_inputs 2109
write_primary_outputs 2110
write_procedure_testbench 2111
write_procfile 2115
write_scan_order 2117

write_scan_setup 2124
write_subchain_setup 2125
write_test_point_dofile 2127
write_visualizer_dofile 2129
write_visualizer_preferences 2131
write_window_contents 2132

Chapter 8
Design Rule Checking 2135

Design Rule Overview 2135
Scan Insertion Checking 2135
Troubleshooting Rules Violations 2136
Design Rules 2147
RAM Rules (A Rules) 2147
A1 2148
A2 2149
A3 2150
A4 2151
A5 2152
A6 2153
A7 2154
A9 2155
A10 2156
A11 2157
A12 2158
A13 2159
A14 2160
A15 2161
A16 2162
BIST Rules (B Rules) 2163
B2 2164
B4 2165
Clock Rules (C Rules) 2166
C1 2170
C2 2175
C3 2177
C4 2184
C5 2191
C6 2193
C7 2195
C8 2197
C9 2199
C10 2201
C16 2202
C17 2204
C19 2205
Scan Cell Data Rules (D Rules) 2207
D1 2208

D2 2213
D3 2214
D4 2215
D5 2216
D6 2218
D7 2220
D8 2221
D9 2223
D10 2225
D11 2227
D12 2229
Extra Rules (E Rules) 2230
E1 2231
E2 2232
E3 2233
E4 2234
E5 2238
E6 2240
E7 2241
E8 2242
E9 2243
E10 2244
E11 2249
E12 2250
E13 2251
E14 2252
EDT Finder Rules (F Rules) 2253
F3 2254
F4 2255
F5 2256
F6 2257
F7 2259
F8 2260
F9 2262
F10 2263
F11 2264
F12 2265
F13 2266
F14 2267
F15 2268
F16 2269
F17 2270
F18 2271
F19 2272
F20 2273
F21 2275
F22 2276
Flattening Rules (FN, FP, and FG Rules) 2279
FN1 2280

FN2 2281
FN3 2282
FN4 2283
FN5 2284
FN6 2285
FN7 2286
FN8 2287
FN9 2288
FP1 2289
FP2 2290
FP3 2291
FP4 2292
FP5 2293
FP6 2294
FP7 2295
FP8 2296
FP9 2297
FP10 2298
FP11 2299
FP12 2300
FP13 2301
FG1 2302
FG2 2303
FG3 2304
FG4 2305
FG5 2306
FG6 2307
FG7 2308
FG8 2309
General Rules (G Rules) 2310
G1 2311
G2 2312
G3 2313
G4 2314
G5 2315
G6 2316
G7 2317
G8 2318
G9 2319
G10 2320
G11 2321
G12 2322
ICL Extraction Rules (I Rules) 2323
I1 2324
I2 2325
I3 2327
I4 2329
I5 2330
I6 2331

ICL Semantic Rules (Semantic Rules) 2333
ICL Semantic Rule 1 2334
ICL Semantic Rule 2 2335
ICL Semantic Rule 3 2336
ICL Semantic Rule 4 2337
ICL Semantic Rule 5 2338
ICL Semantic Rule 6 2339
ICL Semantic Rule 7 2340
ICL Semantic Rule 8 2341
ICL Semantic Rule 9 2342
ICL Semantic Rule 10 2343
ICL Semantic Rule 11 2344
ICL Semantic Rule 12 2345
ICL Semantic Rule 13 2346
ICL Semantic Rule 14 2347
ICL Semantic Rule 15 2348
ICL Semantic Rule 16 2349
ICL Semantic Rule 17 2350
ICL Semantic Rule 18 2351
ICL Semantic Rule 19 2352
ICL Semantic Rule 20 2353
ICL Semantic Rule 21 2354
ICL Semantic Rule 22 2355
ICL Semantic Rule 23 2356
ICL Semantic Rule 24 2357
ICL Semantic Rule 25 2358
ICL Semantic Rule 26 2359
ICL Semantic Rule 27 2360
ICL Semantic Rule 28 2361
ICL Semantic Rule 29 2362
ICL Semantic Rule 30 2363
ICL Semantic Rule 31 2364
ICL Semantic Rule 32 2365
ICL Semantic Rule 33 2366
ICL Semantic Rule 34 2367
ICL Semantic Rule 35 2368
ICL Semantic Rule 36 2369
ICL Semantic Rule 37 2370
ICL Semantic Rule 38 2371
ICL Semantic Rule 39 2372
ICL Semantic Rule 40 2373
ICL Semantic Rule 41 2374
ICL Semantic Rule 42 2375
ICL Semantic Rule 43 2376
ICL Semantic Rule 44 2377
ICL Semantic Rule 45 2378
ICL Semantic Rule 46 2379
ICL Semantic Rule 47 2380
ICL Semantic Rule 48 2381

ICL Semantic Rule 49 2382
ICL Semantic Rule 50 2383
ICL Semantic Rule 51 2384
ICL Semantic Rule 52 2385
ICL Semantic Rule 53 2386
ICL Semantic Rule 54 2387
ICL Semantic Rule 55 2388
ICL Semantic Rule 56 2389
ICL Semantic Rule 57 2390
ICL Semantic Rule 58 2391
ICL Semantic Rule 59 2392
ICL Semantic Rule 60 2393
ICL Semantic Rule 61 2394
ICL Semantic Rule 62 2395
ICL Semantic Rule 63 2396
ICL Semantic Rule 64 2397
ICL Semantic Rule 65 2398
ICL Semantic Rule 66 2399
ICL Semantic Rule 67 2400
ICL Semantic Rule 68 2401
ICL Semantic Rule 70 2402
ICL Semantic Rule 73 2403
ICL Semantic Rule 74 2404
ICL Semantic Rule 75 2405
ICL Semantic Rule 76 2406
ICL Semantic Rule 77 2407
ICL Semantic Rule 78 2408
ICL Semantic Rule 79 2409
ICL Semantic Rule 80 2410
ICL Semantic Rule 81 2411
ICL Semantic Rule 82 2412
ICL Semantic Rule 83 2413
ICL Semantic Rule 84 2414
ICL Semantic Rule 85 2415
ICL Semantic Rule 86 2416
ICL Semantic Rule 87 2417
ICL Semantic Rule 88 2418
ICL Semantic Rule 89 2419
ICL Semantic Rule 90 2420
ICL Semantic Rule 91 2421
ICL Semantic Rule 92 2422
ICL Semantic Rule 93 2423
ICL Semantic Rule 94 2424
ICL Semantic Rule 95 2425
ICL Semantic Rule 96 2426
ICL Semantic Rule 97 2427
ICL Semantic Rule 98 2428
ICL Semantic Rule 99 2429
ICL Semantic Rule 100 2430

ICL Semantic Rule 101 2431
ICL Semantic Rule 102 2432
ICL Semantic Rule 103 2433
ICL Semantic Rule 104 2434
ICL Semantic Rule 105 2435
ICL Semantic Rule 106 2437
ICL Semantic Rule 107 2438
ICL Semantic Rule 108 2439
ICL Semantic Rule 109 2440
ICL Semantic Rule 110 2441
ICL Semantic Rule 111 2442
ICL Semantic Rule 112 2443
EDT Rules (K Rules) 2444
K1 2445
K2 2446
K3 2447
K4 2448
K5 2449
K6 2450
K7 2452
K8 2453
K9 2454
K10 2456
K11 2457
K12 2458
K13 2459
K14 2460
K15 2461
K16 2465
K17 2466
K18 2467
K19 2468
K20 2470
K21 2471
K22 2472
K23 2474
K24 2475
K25 2477
Procedure Rules (P Rules) 2478
P1 2479
P2 2480
P3 2481
P4 2482
P5 2483
P6 2484
P7 2485
P8 2486
P9 2487
P10 2488

P11 2489
P12 2490
P13 2491
P14 2492
P15 2493
P16 2494
P17 2495
P18 2496
P19 2497
P20 2498
P21 2499
P22 2500
P23 2501
P24 2502
P25 2503
P26 2504
P27 2505
P28 2506
P29 2507
P30 2508
P31 2509
P32 2510
P33 2511
P34 2512
P35 2513
P36 2514
P37 2515
P38 2516
P39 2517
P40 2518
P41 2519
P42 2520
P43 2521
P44 2522
P45 2523
P46 2524
P47 2525
P48 2526
P49 2527
P50 2528
P51 2529
P52 2530
P53 2531
P54 2533
P55 2534
P56 2535
P58 2536
P59 2537
P62 2538

P63 2539
P64 2540
P65 2541
P66 2542
P70 2543
P71 2544
P72 2545
P73 2546
P74 2547
P75 2548
P76 2549
P77 2550
P78 2551
P79 2552
P80 2553
P81 2554
P82 2555
P83 2556
P84 2557
P85 2558
P86 2559
P87 2560
P88 2561
P89 2562
P90 2563
P91 2564
P92 2565
P93 2566
Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules) 2567
R1 2568
R2 2569
R3 2570
R4 2571
R5 2572
R6 2573
R7 2574
R8 2575
R9 2577
R10 2579
R11 2580
R12 2581
R13 2582
Scannability Rules (S Rules) 2583
S1 2584
S2 2585
S3 2586
S4 2587
S5 2588
S6 2589

Scan Chain Trace Rules (T Rules) 2590
T1 2591
T2 2592
T3 2593
T4 2594
T5 2595
T6 2596
T8 2597
T9 2598
T10 2599
T11 2600
T12 2601
T13 2602
T14 2603
T15 2604
T16 2605
T17 2606
T18 2607
T19 2608
T20 2609
T21 2610
T22 2611
T23 2612
T24 2613
T25 2614
T26 2616
Power-Aware Rules (V Rules) 2617
V1 2618
V2 2619
V3 2620
V4 2621
V5 2622
V6 2623
V7 2624
V8 2625
V9 2626
V10 2627
V11 2628
V12 2629
V13 2630
V14 2631
V15 2632
V16 2633
V17 2634
V18 2635
V19 2636
V20 2637
V21 2638
Timing Rules (W Rules) 2639

W1 2640
W2 2641
W3 2642
W4 2643
W5 2644
W6 2645
W7 2646
W8 2647
W9 2648
W10 2649
W11 2650
W12 2651
W13 2652
W14 2653
W15 2654
W17 2655
W18 2656
W19 2657
W20 2658
W21 2659
W22 2660
W23 2661
W24 2662
W25 2663
W26 2664
W27 2665
W28 2666
W29 2667
W30 2668
W31 2669
W32 2670
W33 2671
W34 2672
W35 2673
W36 2674
W37 2675
W38 2676
Other DRC Messages 2677

Appendix A
Parameter File Format and Keywords 2679

What is a Parameter File? 2679
Parameter File Syntax 2679
Parameter File Keywords. 2680
A Note about the ALL_* Parameter File Keywords 2683
ALL_EXCLUDE_POWER_GROUND. 2683
ALL_EXCLUDE_UNUSED 2683
ALL_FIXED_CYCLES 2683

ALL_FLATTEN_TIMING 2684
ALL_IDDQ_TESTER_CYCLE. 2684
ALL_MIN_SCAN_LOAD. 2685
ALL_NO_CYCLE_OPT 2685
ALL_NO_LOOP 2686
ALL_NO_PATTERN_TYPE. 2686
ALL_NONSCAN_CONSTANT 2686
ALL_ORIGINAL_INDEX. 2686
ALL_TIME_RESOLUTION 2687
ALL_USE_CYCLE_LOOP 2687
CTL_ONE_PAT. 2687
CTL_ONE_PROTO 2688
CTL_STIL_0 2688
CTL_STIL_1 2689
FTDL_DESIGNER 2689
FTDL_MAX_PAT_BLOCK 2689
FTDL_REVISION 2689
FTDL_TEST_NAME. 2690
FTDL_TEST_NUM 2690
FTDL_TEST_SUFFIX. 2690
FTDL_ZMODE_MES 2690
SIM_ANNOTATE_QUIET 2691
SIM_CHAIN_ERROR. 2691
SIM_CHANGE_PATH 2692
SIM_COMPARE_X. 2692
SIM_DELAY_SCAN_RELEASE 2692
SIM_DIAG_FILE. 2693
SIM_DISTRIBUTED_PAT 2693
SIM_EARLY_RELEASE 2694
SIM_EARLY_RELEASE_TIME 2694
SIM_FILL_BIDISCAN 2695
SIM_FORCE_Z_AT_CYCLE 2695
SIM_INCLUDE 2695
SIM_INSTANCE_NAME 2696
SIM_KEEP_PATH. 2696
SIM_MASK_FILE. 2696
SIM_MIN_NAME_FILE. 2697
SIM_MISCOMPARE_LIMIT 2697
SIM_MODULE_INCLUDE 2698
SIM_MODULE_NAME 2698
SIM_NO_CHAINNAME_FILE 2698
SIM_NO_MEASURE_IN 2698
SIM_PARALLEL_DELAY_SHIFT_CLOCKS 2699
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE. 2699
SIM_POST_SHIFT 2699
SIM_PRE_SHIFT. 2700
SIM_PRECISION 2700
SIM_SERIAL_CHAIN 2700
SIM_SHIFT_DEBUG 2701

SIM_SIGNAL_SPY 2701
SIM_STATUS_MSG 2702
SIM_TIMEPLATE_COMM 2702
SIM_TOP_NAME 2702
SIM_VECTOR_COMM 2703
SIM_VECTYPE_SIGNAL 2704
STIL_2005_SCAN 2705
STIL_CHAIN_CELLS 2705
STIL_CHANNEL_CELLS 2706
STIL_FULL_CHAIN 2706
STIL_MIN_QUOTE 2706
STIL_NESTED_MACRO 2707
STIL_NO_SCANSTRUCTURE 2707
STIL_NO_SCANX 2707
STIL_NOMEASURE_CLOCK 2707
STIL_PAT_ANN 2708
STIL_PAT_CMT 2708
STIL_PAT_LAB 2709
STIL_QUOTE_ALL 2709
STIL_SCAN_ANN1 2709
STIL_STRUCTURAL 2710
STIL_TI_TITLE 2712
STIL_TRIM_PULSE 2713
STIL_VECTOR_ANN 2713
STIL_VERG_ESC 2714
TITDL_CHAIN_SETTYPE 2715
TITDL_CUSTOMER 2715
TITDL_GROUP_TIMING 2715
TITDL_KEEP_SCANOUT 2715
TITDL_LIBRARY 2716
TITDL_PARTNUM 2716
TITDL_PSEUDO_PREFIX 2716
TITDL_REVISION 2717
TITDL_SETNAME 2717
TITDL_SETTYPE 2718
WGL_ADD_LASTX 2718
WGL_ALT_BIDI 2718
WGL_ALT_VECT_ANN 2719
WGL_EDGE_STROBE 2719
WGL_FULL_CHAIN 2721
WGL_FULL_SCANGROUP 2722
WGL_GROUP_PIN 2722
WGL_INV_SC 2723
WGL_NO_SCANX 2723
WGL_NOMEASURE_CLOCK 2723
WGL_ONE_ILLINOIS 2724
WGL_PATTERN_NAME 2724
WGL_TRIM_LEADING_P 2724
WGL_TRIM_PULSE 2725

WGL_VECTOR_ANN 2725
WGL_VERG_ESC. 2726
WGL_VTRAN_PADSC 2726

Appendix B
Getting Help 2727

Documentation. 2727
Mentor Graphics Support. 2728

Third-Party Information

End-User License Agreement

List of Figures

Figure 3-1. Tessent Shell Design Object Model 58
Figure 5-1. IJTAG Network Example Schematic 118
Figure 5-2. DesignInstance Example Schematic 122
Figure 5-3. HostScanInterface/Interface Example Schematic 127
Figure 5-4. DataInPorts Example Schematic 131
Figure 5-5. ScanMux Internal Schematic When Used on a DR Scan Path 135
Figure 5-6. ScanMux Internal Schematic When Used on a TAP Scan Path 136
Figure 5-7. ScanMux Example Schematic 139
Figure 5-8. SIB Internal Schematic 144
Figure 5-9. Added Scan Isolation Logic for SIB(sti) 145
Figure 5-10. SIB Example Schematic 148
Figure 5-11. TAP FMS Diagram 153
Figure 5-12. TAP Block Diagram 154
Figure 5-13. Example Multi-TAP Configuration Schematic 156
Figure 5-14. Update Stage Timing Diagram 173
Figure 5-15. Detailed View of TDR Schematic 176
Figure 5-16. DataOutPort Connection Multiplexing 181
Figure 5-17. Example Option panel where blank spaces were added 203
Figure 7-1. Merging Patterns for Multiple Views of a Core at the Top Level 435
Figure 7-2. Merging Patterns for Top Level and Core at the Top Level 437
Figure 7-3. Example of a False Path to be Defined 456
Figure 7-4. Example Circuit for Defining False Paths 457
Figure 7-5. Logical Connection Attributes 478
Figure 7-6. Logical Connection Example 479
Figure 7-7. MISR placement 485
Figure 7-8. analyze_simulation_mismatches Example 604
Figure 7-9. Extra Layout Hierarchy Example 664
Figure 7-10. Extra Design Hierarchy Example 665
Figure 7-11. Selecting Two Nodes in the Configuration Tree 854
Figure 7-12. Various Components of a Configuration Element 855
Figure 7-13. Circuit Illustrating the Usage of the get_fanins Command 891
Figure 7-14. Circuit Illustrating the Usage of the get_fanouts Command 894
Figure 7-15. iApply Time Frame 1002
Figure 7-16. Inverter Interception 1032
Figure 7-17. Intercepting an Input Pin From Inside An Instance 1075
Figure 7-18. Intercepting an Output Port From Inside the Current Design 1076
Figure 7-19. Intercepting an Input Pin From Outside An Instance 1077
Figure 7-20. Intercepting an Output Pin From Outside An Instance 1078
Figure 7-21. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 0 1086
Figure 7-22. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 0 1086

Figure 7-23. Scan Frame Timing with tck_ratio = 4 and TCK Off Value of 0 1087
Figure 7-24. Scan Frame Timing with tck_ratio >= 8 and TCK Off Value of 0 1087
Figure 7-25. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 1 1088
Figure 7-26. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 1 1088
Figure 7-27. Scan Frame timing with tck_ratio = 4 and TCK Off Value of 1 1089
Figure 7-28. Scan Frame timing with tck_ratio >= 8 and TCK Off Value of 1 1089
Figure 7-29. Case Analysis Example 1157
Figure 7-30. set_clock_off_simulation Off and set_split_capture_cycle Off 1514
Figure 7-31. set_clock_off_simulation On and set_split_capture_cycle Off 1514
Figure 7-32. set_clock_off_simulation Off and set_split_capture_cycle On 1514
Figure 7-33. set_clock_off_simulation On and Set Split Capture On 1515
Figure 7-34. Global versus Local Bus Simulation Analysis Example 1588
Figure 7-35. False Path Intersection and Effect Cones 1754
Figure 7-36. Scan Cell Used as a Control and Observe Point 1917
Figure 7-37. Power Isolation Cells in Design Block with Three Power Domains 1918
Figure 7-38. Dedicated Wrapper Cell Insertion When Power Isolation Cells Present 1919
Figure 7-39. I/O Identification Default Tracing Mode 1991
Figure 7-40. All Input Logic Tracing 1991
Figure 7-41. Control and Observe Point Insertion 1992
Figure 7-42. Debug Window After add_display_instance Command 2020
Figure 7-43. Result of trace_flat_model While Tagging Multiplexers 2021
Figure 8-1. Example of Design Level 2141
Figure 8-2. Example of Primitive Level 2142
Figure 8-3. Data Reported for a Specific Gate 2144
Figure 8-4. Clock Cycle Terminology 2166
Figure 8-5. Example Clock Cone 2167
Figure 8-6. Example Effect Cone 2167
Figure 8-7. Pin in Both Cones 2168
Figure 8-8. DFTVisualizer Debug Window View Showing Clock Cones 2168
Figure 8-9. C1 Violation Example 2171
Figure 8-10. C2 Rule Example Circuit 2176
Figure 8-11. C3 Violation Example 2178
Figure 8-12. Example Timing That Allows Sink to Capture Source’s New Data 2178
Figure 8-13. C4 Violation Example 2185
Figure 8-14. Example Where Actual Behavior Differs from Tool’s Prediction 2185
Figure 8-15. C5 Rule Example Circuit 2192
Figure 8-16. C6 Rule Example Circuit 2194
Figure 8-17. C7 Rule Example Circuit 2196
Figure 8-18. C8 Rule Example Circuit 2198
Figure 8-19. C9 Rule Example Circuit 2200
Figure 8-20. C10 Rule Example Circuit 2201
Figure 8-21. Rule D10 Violation Example 2225
Figure 8-22. Rule D11 Violation Example 2227
Figure 8-23. Example E4 Violation Trace in Debug Window of DFTVisualizer 2235
Figure 8-24. Example E10 Violation Trace in Debug Window of DFTVisualizer 2246

Figure 8-25. Blocked Decompressor Input 2256
Figure 8-26. Blocked Channel Input to Decompressor 2257
Figure 8-27. Output Channel and Compactor Connection Blocked 2260
Figure 8-28. Blocked Scan Chain Path 2262
Figure 8-29. analyze_drc_violation F22 in Test Structures Window 2277
Figure 8-30. Callout Text for reg_lfsm_vec_lockup_3 2277
Figure 8-31. analyze_drc_violation F22 in Debug Window 2278
Figure 8-32. Failing identification 2455
Figure 8-33. Common cell structure 2455
Figure 8-34. Failing identification 2455
Figure 8-35. Example S3 Rule Violation. 2586
Figure 8-36. Example of T25 Violation. 2614
Figure 8-37. Transparent Capture Handling Analysis Messages. 2677
Figure 8-38. RAM Summary Results and Test Capability Messages 2678

List of Tables

Table 1-1. Tessent Point Tools 40
Table 2-1. Shell Command Summary 47
Table 3-1. Built-in Primitives and Primitive Port Names 82
Table 4-1. Commands for Manipulating Attributes 103
Table 4-2. Attribute Value Types 104
Table 4-3. Posix Extended Regular Expression Metacharacters 107
Table 4-4. Special Characters in Regular Expressions 109
Table 4-5. Special Characters Set in Regular Expression 109
Table 5-1. Syntax Conventions for Configuration Files 113
Table 6-1. Syntax Conventions for ICL Files 224
Table 7-1. Conventions for Command Line Syntax 341
Table 7-2. Command Summary 342
Table 7-3. Example Cell Value Changes with Different Constraints for Scan Patterns 406
Table 7-4. Example Scan Cell Value Changes with Different Constraints for Chain Patterns 407
Table 7-5. How the DFTVisualizer Debug Window Displays Instances 440
Table 7-6. Logical Connection Attributes 477
Table 7-7. Subchain Types 553
Table 7-8. Fault Class Codes and Class Names 730
Table 7-9. Types of Configuration Elements 857
Table 7-10. Available Partitions for Introspection 860
Table 7-11. Events of type “vector” 964
Table 7-12. Events of type “scan” 965
Table 7-13. Events of type “reset” 967
Table 7-14. Events of type “loop” 967
Table 7-15. Events of type “goto_state” 967
Table 7-16. Events of type “nop” 968
Table 7-17. Interception based on node type 1027
Table 7-18. Allowed Cell Function Names 1030
Table 7-19. RunLoop Scaling Algorithm 1047
Table 7-20. Mask File Statements 1147
Table 7-21. Named Capture Procedure Merging Under Different Conditions 1151
Table 7-22. Clock Compatibility Summary 1233
Table 7-23. Available Partition for Introspection 1247
Table 7-24. Default Netlist Information Reported for Instances 1269
Table 7-25. Available Information Displayed and Arguments 1272
Table 7-26. Fault Class Codes and Names 1325
Table 7-27. Name Conventions Used by report_faults -Cell_name 1327
Table 7-28. Reportable Gate Types 1352
Table 7-29. Tessent FastScan and Tessent TestKompress Clock Port Categories 1353

Table 7-30. Notation Used in report_patterns Display 1429
Table 7-31. Default report_processors Information Display 1449
Table 7-32. Additional Data Displayed by -Verbose 1450
Table 7-33. Output Format Codes 1484
Table 7-34. Format Codes 1496
Table 7-35. Reason for Failed Identification 1533
Table 7-36. Untestable Faults that are Reclassified by reset_au_faults 1541
Table 7-37. Default Clock Compatibility Summary at a Clock Domain Boundary 1622
Table 7-38. Available Partition for Introspection 1630
Table 7-39. Contexts Available by License 1638
Table 7-40. Predefined Simulation Contexts 1648
Table 7-41. Pattern Sources 1742
Table 7-42. Lockup Cell(s) Used Between Different Clock/Edge Transitions 1806
Table 7-43. Instance Type Prefix Defaults 1846
Table 7-44. WIRE Bus Contention Truth Table 1849
Table 7-45. AND Bus Contention Truth Table 1849
Table 7-46. OR Bus Contention Truth Table 1849
Table 7-47. Keywords for Pattern Types 1865
Table 7-48. Mask File Statements 1878
Table 7-49. Fault Classes for Relevant Coverage 1921
Table 7-50. Fault Sub-Classes for Relevant Coverage 1921
Table 7-51. Default Scan Enable Signal Names 1924
Table 7-52. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch 1969
Table 7-53. Single Module Per File Naming Convention 2058
Table 7-54. Name Conventions Used by write_faults -Cell_name 2082
Table 8-1. DRC Violation Messages 2137
Table 8-2. Clocking that Can Result in a C3 Signal Race 2177
Table 8-3. Clocking that Can Result in a C4 Signal Race 2184
Table 8-4. ICL Extraction DRCs 2323
Table 8-5. Port Connections 2327
Table 8-6. Port Types and Required Input Binding 2382
Table A-1. Vector Types 2703
Table A-2. Translation Table 2704
Table A-3. STIL Special Keywords 2713
Table A-4. STIL Pattern Types 2714
Table A-5. STIL Vector Types 2714

Chapter 1
Introduction

This manual describes the application commands for Tessent Shell, as well as the following products: Tessent FastScan, Tessent TestKompress, Tessent Diagnosis, Tessent IJTAG, Tessent DesignEditor, and Tessent Scan (formerly DFTAdvisor).

- Tessent FastScan is a comprehensive combinational Automatic Test Pattern Generation (ATPG) system optimized for full-scan designs. It offers high speed and accurately measured high test coverage to guarantee your product quality and reliability.
- Tessent TestKompress uses patented scan EDT™ (Embedded Deterministic Test) technology and many of the industry leading ATPG features of Tessent FastScan, to reduce scan data volume and test time by many times over Tessent FastScan ATPG. It is for users who need greater reductions in ATE memory and time requirements than are available with any other full-scan ATPG tool.
- Tessent Diagnosis is a sophisticated diagnosis tool used at different stages of test to help characterize and locate defects in manufactured chips. It is for users who need to shorten the time it takes to perform yield ramp, yield learning and customer return analysis. The unique advanced diagnosis capabilities it provides are also available in Tessent TestKompress.
- Tessent IJTAG is Mentor Graphics implementation of the IEEE P1687 (IJTAG) and supports reading Instrument Connectivity Language (ICL), performing automatic retargeting of IJTAG-specific PDL commands such as iWrite and iRead, and writing out the retargeted PDL patterns in many different ATE tester specific formats and as Verilog HDL simulation test benches. See the *Tessent IJTAG User's Manual* for complete information.
- Tessent DesignEditor is a less expensive product that provides access only to design introspection and editing features. This product only provides access to the dft context (commands labeled "dft" or "all contexts").
- Tessent Scan is the Mentor Graphics internal scan synthesis tool that identifies and inserts scan and test circuitry into your design. For more information, refer to the *Tessent Scan and ATPG User's Manual.*
- DFTVisualizer which is built in to all of these tools is a schematic and data viewing utility. DFTVisualizer adds the ability to graphically investigate and interact with designs, making it easier to debug problems. For more information, see "DFTVisualizer" in the *Tessent Shell User's Manual*.

In Tessent Shell, the term “context” refers to a broad category of functionality that often corresponds to a specific product and a license feature, such as Tessent FastScan. By setting the context, you indicate the type of task you want Tessent Shell to perform.

Table 1-1 lists the Tessent products supported by this manual and shows how they map to Tessent Shell contexts. For more information about Tessent Shell contexts, refer to the “Context and System Modes” section in the *Tessent Shell User’s Manual*.

Table 1-1. Tessent Point Tools

Product or Tool	Tessent Shell contexts	For more information
Tessent FastScan Tessent TestKompress (ATPG operations)	patterns -scan	See the “Generating Test Patterns” chapter in the *Tessent Scan and ATPG User’s Manual*.
Tessent TestKompress (IP insertion)	dft -edt	See the *Tessent TestKompress User’s Manual*.
Tessent IJTAG	patterns -ijtag	See the *Tessent IJTAG User’s Manual*.
Tessent DesignEditor	“dft” or “all contexts”	See “Tessent DesignEditor” on page 43.
Tessent Scan (formerly DFTAdvisor)	dft -scan	See the “Inserting Internal Scan and Test Circuitry” section in the *Tessent Scan and ATPG User’s Manual*.
Tessent Diagnosis	patterns -scan_diagnosis	See the *Tessent Diagnosis User’s Manual*.

Overview

This section summarizes the features and inputs/outputs of the products documented in this manual.

- "Tessent FastScan and Tessent TestKompress" on page 41
- "Tessent Scan" on page 44
- "Tessent DesignEditor" on page 43
- "Tessent Diagnosis" on page 45
- "Tessent IJTAG" on page 45

Tessent FastScan and Tessent TestKompress

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

Both products share the following features:

- Can be used within a Mentor Graphics flow or as a point tool in other design flows.
- Contain an internal high-speed fault simulator.
- Produce a number of standard test pattern data formats.
- Contain a powerful design rules checker.
- Read most standard gate-level netlists.
- Produce very high coverage test pattern sets for full-scan and scan-sequential designs. Scan-sequential designs contain well-behaved sequential scan circuitry, including non-scan latches, sequential memories, and limited sequential depth.
- Contain functionality for handling embedded RAM and ROM.
- Support the same pattern types and fault models.
- Provide the same test coverage.
- Provide the same diagnostics.
- Contain the same command set, with the exception that Tessent TestKompress contains additional commands for EDT IP creation and use.

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

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

Tessent FastScan and Tessent TestKompress Inputs

Tessent FastScan and Tessent TestKompress have the following inputs:

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

Tessent TestKompress has the following inputs:

- EDT IP Creation Phase:
  - **Design** — This must be a gate-level netlist with scan. The supported netlist format is Verilog.
  - **Test Procedure File** — This file defines the operation of the scan circuitry in your design. You can generate the file by hand or by using the write_atpg_setup command in Tessent Scan. For more information on test procedure files, refer to the "Test Procedure File" chapter of the *Tessent Shell User's Manual*.
  - **Library** — This file contains model descriptions for all library cells used in your design.
- EDT Pattern Generation Phase:
  - **Design** — This must be a scan-inserted, gate-level netlist with EDT circuitry. The supported netlist format is Verilog.
  - **EDT Dofile** — Created by Tessent TestKompress in the IP Creation Phase, this file contains setup commands for the Pattern Generation Phase.
  - **EDT Test procedure file** — Created by Tessent TestKompress in the IP Creation Phase, this file contains test procedure steps for the Pattern Generation Phase.
  - **Library** — This file contains model descriptions for all library cells used in your design.

Tessent FastScan and Tessent TestKompress Outputs

Tessent FastScan and Tessent TestKompress produce the following outputs:

- **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test

pattern formats, refer to the write_patterns command reference page within this manual, or the "Saving the Test Patterns" section in the *Tessent Scan and ATPG User's Manual*.

- **ATPG Information Files** — These files contain session information that you can save using various commands.
- **Fault List** — This is an ASCII file that contains internal fault information in the standard Mentor Graphics fault format.

Tessent TestKompress produces the following outputs:

- EDT IP Creation Phase:
  - **EDT IP Files** — These files contain the EDT IP described in Verilog or VHDL RTL, as well as connectivity and a black box representation of the core design.
  - **Design Compiler Synthesis Script** — This is a synthesis script, typically used as a template for synthesizing the EDT IP into the design.
  - **EDT Dofile** — This file contains setup commands for the EDT Pattern Generation Phase.
  - **EDT Test Procedure File** — This file contains test procedure steps for the Pattern Generation Phase.
  - **Bypass Files** — These files consist of a dofile and a test procedure file for optionally performing Tessent FastScan ATPG on the design with EDT.
- EDT Pattern Generation Phase:
  - **Test Patterns** — This file set contains test patterns in one or more of the supported simulator or ASIC vendor pattern formats. For more information on the available test pattern formats, refer to the write_patterns command description in this manual.

Tessent DesignEditor

Tessent DesignEditor features contain a subset of Tessent Shell features.

Tessent DesignEditor features include the following:

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

Tessent DesignEditor Inputs

Tessent DesignEditor has the following inputs:

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

- **Library** — This file contains model descriptions for all library cells used in your design.

Tessent DesignEditor Outputs

Tessent DesignEditor produces the following outputs:

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

Tessent Scan

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

Specifically, Tessent Scan supports the following functionality:

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

Tessent Scan Inputs

Tessent Scan has the following inputs:

- Design Netlist — In Verilog format.
- Test Procedure File — Required if you have existing scan circuity in your design. The test procedure file defines the operation of existing scan circuitry. See also "Specifying Existing Scan Information."
- DFT Library — Contains the model descriptions for all library cells used in your design, along with the model descriptions for all the scan replacement cells.
- **Command Dofile** — A text file containing a list of Tessent Scan commands. You can also enter these commands interactively.

Tessent Scan Outputs

Tessent Scan produces the following outputs:

- Design Netlist — A scan version of your design netlist that can be in Verilog format.

- ATPG Setup Files — The test procedure file, which defines the operation of the scan circuitry in your design, and a dofile for setting up the design and scan circuitry information for ATPG operations.

Tessent Diagnosis

Tessent Diagnosis-specific features include the following:

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

Tessent Diagnosis Inputs

Tessent Diagnosis has the following inputs:

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

Tessent Diagnosis Outputs

Tessent Diagnosis produces the following outputs:

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

Tessent IJTAG

Tessent IJTAG is Mentor Graphics implementation of the IEEE P1687 (IJTAG) for PDL command retargeting and ICL extraction. For complete information, refer to the *Tessent IJTAG User's Manual*.

Tessent IJTAG Inputs

Tessent IJTAG produces the following input:

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

Tessent IJTAG Outputs

Tessent IJTAG produces the following outputs:

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

Optimization of Adobe Reader

You can optimize your ability to navigate through the manuals by customizing your Adobe Reader user settings to display the and icons. These Previous View and Next View icons allow you to retrace your viewing history. This optimization is particularly useful for navigating the configuration data syntax presented in the Configuration-Based Specification chapter of this manual.

You can enable these icons by clicking the right mouse button on the tool banner and selecting **Page Navigation > Previous View | Next View** from the popup menu. After you enable these icons, they will be persistent in your personal Adobe Reader environment.

Chapter 2
Shell Commands

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

Shell Command Descriptions

The notational conventions used to describe the shell commands are the same as those used in other parts of the manual. Do not enter any of the special notational characters (such as, {}, [], or |) when typing the command. For a complete description of the notational conventions used in this manual, refer to "Command Line Syntax Conventions" on page 341.

Table 2-1 contains a summary of the shell commands described in this chapter.

Table 2-1. Shell Command Summary

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

create_skeleton_design

Usage

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

Description

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

This tool requires a special design skeleton input file that you create manually. For more information about this file, see "Integrating Compression at the RTL Stage" in the *Tessent TestKompress User's Manual.*

Arguments

- -i *skeleton_design_input_file*

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

- -o *output_file_prefix*

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

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

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

- -simulation_library

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

- -design_interface *file_name*

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

- -module_name *module_name*

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

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

- -help

  An optional switch that lists all the invocation switches.

- -verbose

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

stil2mgc

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

Usage

stil2mgc {-**STil** ***stil_filename*** [-**TPf** ***tpf_filename***] [-**DOfile** ***dofile_name***]
[-FLex_dofile *dofile_name*]
[-ALias Min | All] [-CApture NOne | Single | NAmed]
[-LOgfile *logfile_name* [-REplace]]} |
[-help | -Usage | -MANual | -Version] [-Verbose]

Description

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

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

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

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

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

Arguments

- **-STil** ***stil_filename***

  A required switch and string pair that specifies the name of the input STIL Procedure File (SPF).

- **-TPf** ***tpf_filename***

  An optional switch and string pair that specifies the name of the test procedure file to generate. If not specified, the *tpf_filename* will be named by appending “.proc” to the STIL Test Procedure file name.

- **-DOfile *dofile_name***

  An optional switch and string pair that specifies the name of the dofile to generate. If not specified, the *dofile_name* will be named by appending ".dof" to the STIL Test Procedure file name.

- -FLex_dofile *dofile_name*

  An optional switch and string pair that produces an additional dofile targeted at the FlexTest tool.

- -ALias Min | All

  An optional switch and literal pair that produces the minimum required alias statements in the MGC Procedure file or to produce alias statements for all signal groups found in the STIL Test Procedure file. The default is to produce the minimum required alias statements.

- -CApture NOne | Single | NAmed

  An optional switch and literal pair that adds information to the MGC Procedure file. This switch will either produce no capture procedures, a single default capture procedure, or multiple named capture procedures for each capture Macro found in the STIL Test Procedure File. The default is to produce no capture procedures.

- -LOgfile *logfile_name*

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

- -REplace

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

- -help

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

- -Usage

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

- -MANual

  An optional switch that opens the Tessent Documentation System.

- -Version

  An optional switch that displays the version of the **stil2mgc** tool that you currently have available.

Examples

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

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

tessent

Usage

tessent {-Shell
| {-DIAGSERVER *diagnosis_arguments*}
| {-YIELDINSIGHT [-NOGui]}
| {-Siliconinsight {-Desktop | -Ate}}
[-DOFile *dofile_name*... [-HIStory]]
[-Arguments {*name=value*}...]
[-LOGfile *logfile_name* [-REPlace]]
[-LICense_wait {*minutes* | NONE | UNLimited}]
[-underscore_commands_only]
| [-Help | -Usage | -Manual | -Version]

Description

Invokes a Tessent tool in a command-line session.

Arguments

Tessent Shell Arguments

- -Shell

  A required switch for invoking Tessent Shell. For more information, refer to the *Tessent Shell User's Manual*.

- -DOFile *dofile_name*... [-HIStory]

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to execute upon invocation. As the tool is processing each dofile, the Tcl variable "tessent_user_argv0" is assigned the name of the current dofile. -HIStory is an optional switch that adds dofile commands to the command line history list. The tool ignores the -History switch if you issue it without the -Dofile switch.

- -Arguments {*name=value*}...

  An optional switch and repeatable name/value pair that specifies variable names and values in a list named "tessent_user_arg." No spaces are allowed between the name, the equal sign, and the value. This list can later be accessed by a dofile for the purpose of controlling parameters in the dofile. For more information, refer to "Example 2" on page 55.

Tessent Diagnosis Server Arguments

-DIAGSERVER {[-DAEmon [*daemon_id*]] -DOFile *dofile_name*} [-HDB *history_database_name*]
| [{-DSTATUS | -DEXIT | -DTERMINATE} *daemon_id* | [-DLIST]]

- -DIAGSERVER

  A required switch for invoking Tessent Diagnosis server. See "Running Tessent Diagnosis Server" in the *Tessent Diagnosis User's Manual*.

- -DOFile *dofile_name*... [-HIStory]

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to execute upon invocation. -HIStory is an optional switch that adds dofile commands to the command line history list. The tool ignores the -History switch if you issue it without the -Dofile switch.

- -DAEmon *daemon_id*

  An optional switch that specifies invoking the Tessent Diagnosis server in daemon mode. The tool automatically assigns a process ID. Using the optional *daemon_id*, you can specify a process ID for the daemon.

- -HDB *history_database_name*

  An optional switch and string pair that creates a server history SQL database. If the named database already exists, then Tessent Diagnosis server appends the database during subsequent diagnosis runs.

- -DSTatus *daemon_id*

  An optional switch and integer pair that queries the status of a daemon you identify with *daemon_id* and displays the status of the monitors and analyzers.

- -DEXit *daemon_id*

  An optional switch and integer pair that instructs the daemon to exit after the tool finishes any running diagnosis jobs.

- -DTErminate *daemon_id*

  An optional switch and integer pair that instructs the daemon to immediately exit without finishing any running diagnosis jobs.

- -DLIst

  An optional switch that lists running daemons, including the id, host, and logfile for each daemon.

Tessent YieldInsight Arguments

-YIELDINSIGHT [-NOGui]

- -YIELDINSIGHT

  A required switch for invoking Tessent YieldInsight. See the *Tessent YieldInsight User's Manual* for complete information.

- -NOGui

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

- -DOFile *dofile_name*... [-HIStory]

  An optional switch and repeatable string pair that specifies the name of one or more dofiles to execute upon invocation. -HIStory is an optional switch that adds dofile commands to the

command line history list. The tool ignores the -History switch if you issue it without the -Dofile switch.

Tessent SiliconInsight Arguments

-Siliconinsight {-Desktop | -Ate}

- -Siliconinsight

  A required switch for invoking Tessent SiliconInsight.

- -Desktop | -Ate

  A required literal choice that specifies which Tessent SiliconInsight tool to invoke.

  Choose one of the following options:

  -Desktop — Invoke Tessent SiliconInsight Desktop.

  -Ate — Invoke Tessent SiliconInsight ATE.

- -DOFile *dofile_name*

  An optional switch and string pair that specifies the name of a dofile to execute upon invocation.

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

Common Arguments Regardless of Mode

- -LOGfile *logfile_name*

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

- -REPlace

  An optional switch that overwrites the -Logfile *logfile_name* if a log file of the same name already exists.

- -LICense_wait *minutes* | NONE | UNLimited

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

  Choose one of the following options:

  minutes — A positive integer that specifies the number of minutes to wait for the license.

  NONE — A literal that directs the Tessent tool to exit immediately if no license is available.

  UNLimited — A literal that directs the Tessent tool to wait with no time limit for a license. This is the default.

- -underscore_commands_only

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

- -Help | -Usage

  An optional switch that lists the switches for the tessent command.

- -Manual

  An optional switch that opens the Tessent Documentation System.

- -Version

  An optional switch that displays the version of the tessent tool you specify.

Example 1

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

tessent -shell -logfile myLogfile -replace -dofile myDofile

Example 2

The following example invokes Tessent Shell and runs a dofile named dofile2. The command line also specifies two variable/value pairs that are placed in a Tcl list named “tessent_value_arg” and thus made available to the dofile.

% **tessent -shell -dofile dofile2 -arguments depth=2 output_dir=outDir**

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

```
# dofile2
# Set the default values of the control parameters
array set params {
   depth 0
   max_nodes -1
   output_dir ""
}

array set params $tessent_user_arg

# Use some params
if { $params(depth) > 0} {
  # do whatever needs to be done in this case…
}

if {  ! [file isdirectory $params(output_dir)] } {
  file mkdir $params(output_dir)
}
```

Example 3

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

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

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

set_transcript_style off
puts "$tessent_user_arg"

# add dofile commands here

exit
```

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

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

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

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

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

Chapter 3
Data Models

Introduction

Tessent Shell has an internal database that is organized into data models which are broad categories of design information (such as hierarchical design data). Each of the data models contains one or more types of design objects (such as pins and modules), and each design object has a list of attributes (such as an ID or bit-width of a bus).

For additional information about the design data models and how to work with them, refer to "Design Data Model, Attributes, and Introspection" in the *Tessent Shell User's Manual*.

You can access the pre-defined attributes on those design objects using the *_attribute* tool commands listed in Table 4-1 on page 103. Those commands also provide a way for you to register new user-defined attributes for those design objects in the database.

The following sections describe the various data models, the types of objects they can contain, and the attributes associated to the objects:

- "Hierarchical Design Data Model" on page 58
- "Flat Design Data Model" on page 80
- "ICL Data Model" on page 87

Hierarchical Design Data Model

The hierarchical design data model contains five different object types: module, instance, port, pin, and net. As suggested by Figure 3-1, instances, pins, and nets are hierarchical objects that exist relative to the current design. Ports and modules are non-hierarchical objects.

The *current design* is the design specified by the set_current_design command.

Figure 3-1. Tessent Shell Design Object Model

Hierarchical Object Types

The six hierarchical design object types are described on the following reference pages: Module, Instance, Port, Pin, Net, and Pseudo_port.

Reference information about attributes is provided in section "Attributes" on page 103.

Module

Description

A module has a simple non-hierarchical name. A module definition can come from a Verilog module, a VHDL entity, a Tessent library model, or it can be a built-in primitive.

The module object type interacts with attributes as follows:

- Attributes are automatically visible as read-only attributes on instances if the same attribute is not also registered on the instance object type. Such attributes remain stored on the module definition and not on the instance itself
- Attributes with the applies_to_child_instances option set to On are visible to all child instances below the module.

Built-In Attributes

The module object type has the following attributes:

cell_type	String that reports characteristics of the cell. This attribute is initially null and is populated by the set_current_design command with information extracted from the cell library files. For example, any cell described as a pad in the library will have cell_type set to “pad”. If your cell library includes the scan mapping information, the non-scan flip-flop will be identified with a nonscan_dff type and the scan equivalent flip-flops will have mux_scan_dff type. See the read_cell_library command description for information on how the cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command if your cell library file only includes the ATPG cell modeling information. *Tessent Cell Library Manual*
file_path_name	String that specifies the pathname to the file where a given module is defined. This attribute is null for created modules or modules without a definition.
has_no_definition	Boolean that is true when a module does not have a definition, but a blackbox model was created during elaboration from its instantiation. You cannot have this type of module before creating the flat module unless you identify the module as a blackbox using the add_black_boxes command.
ijtag_is_icl_module	Boolean that identifies a design module as an ICL module. This attribute is only set in the patterns -ijtag context.
instrument_name	String that is set to the ICL module name it matches when its ijtag_is_icl_module attribute is true. The default is null.

is_created	Boolean that specifies whether the module was created using the create_module command.
is_hard_module	Boolean that specifies the module cannot be edited when set to true. This attribute is automatically set to true for all modules with type = “cell” or “primitive”. This attribute is set for design modules in the dofile using the following command: set_attribute_value -name is_hard_module. This property has applies_to_child_instances set to On by default.
is_incomplete	Boolean that specifies that the module cannot be edited and/or written out when set to true. This attribute is set to true when the module was read with the read_verilog or read_vhdl -blackbox commands, or when the module includes RTL constructs that are not supported by the set_context -no_rtl option. The is_hard_module attribute automatically returns true for any module with the is_incomplete attribute set to true. To write out a module with only its port and parameter definition, read it in normally and write it back out using the write_design -interface option.
is_modified	Boolean that specifies that a module read in from a file was edited during the insertion mode. This attribute is set as soon as the module is modified and remains set until the insertion state is left. See the get_modules command description for usage examples.
is_physical_module	Boolean that specifies that the module is a physical block. It gets set during design elaboration if the module is found in an open TSDB and its level was defined as chip or physical_block when its TSDB was created.
is_power_always_on_cell	Boolean that is set to true when the module is specified as an always_on cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_isolation_cell	Boolean that is set to true when the module is specified as an isolation cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_level_shifter_cell	Boolean that is set to true when the module is specified as a level shifter cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.
is_power_retention_cell	Boolean that is set to true when the module is specified as a retention cell when power data is read using the read_upf or read_cpf commands. By default, this attribute is false.

is_unsaved	Boolean that specifies that a module that was either modified or created within the insertion mode of the dft context was not yet written out. This attribute is set as soon as the module is modified or created and is reset as soon as the module is saved using the write_design command. See the get_modules command description for examples of its usage.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
language	String that specifies the supported library language. Set to “tessent_lib” for Tessent library models. Set to “verilog” for Verilog modules and Verilog primitives. Set to “vhdl” for VHDL entities.
library_name	String that specifies the logic name of the library containing the module definition. All commands that accept a module name look for the module in the default library as specified by the set_logical_design_libraries command. You can use the get_modules command to access modules from different libraries. Libraries are only supported in -rtl sub-contexts. In -no_rtl contexts, one library exists and it is named “work”.
line_number	Integer that specifies a line number in the file where a given module is defined. This number is 0 for created modules and for modules without a definition.
master_name	String that specifies the real name of the module as it was defined independently of the parameterized view.
name	String that specifies the name of the module. Modules are defined from the following sources: • read_verilog files • read_vhdl files • create_design and copy_design commands • read_cell_library files • primitives have built-in names For parameterized modules, a unique name is created for each parameterized view by appending to the master_name the string @PV# where # is an integer starting from 1 and incremented for each parameterized view. Use the get_module_parameter_list and get_module_parameter_value commands to introspect the parameter values.
object_type	Returns the object data type “module”.
power_domain_name	Returns the name of the power domain associated to the current design. This attribute is null for all other modules. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.

preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins on all instances of the module will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
synthesize_before_analysis	Boolean that specifies that the given module is to be synthesized when going to analysis mode. During ICL extraction, the combinational path between the IJTAG port of the modules having a corresponding ICL module is traced and any module encountered along the way is synthesized. If you use a test_setup procedure during ICL extraction, you must set this attribute on all RTL modules which must be synthesized in order for the test setup simulation to propagate its value to the entire ICL network as the pretracer does not traverse sequential cells. The quick synthesis engine uses a cell library defined in the build tree at the given location "$TESSENT_HOME/lnx-x86/lib64/hdleng/lib". Both Tessent and Verilog cell library models are supplied, namely "dft_sim.atpg" and "dft_sim.v". The tessent libraries are automatically loaded when synthesis is used.
type	Set to "cell" for the following modules: • Models read in with read_cell_library. • Verilog modules surrounded by the `celldefine/`endcelldefine directives and read in with read_verilog. Set to "primitive" for the following modules: • Verilog primitives used in the design. Set to "design" for all other modules.
vhdl_architecture	String that reports the architecture of vhdl modules. It is null for modules of other languages. All commands that accept a module name return the vhdl entity associated with the default architecture which is the last compiled version. Use get_modules to access modules with the non default architecture.

Inheritance

Built-in inherited attributes:

- None

Related Commands

get_attribute_list	get_name_list
get_attribute_value_list	get_pins
get_common_parent_instance	get_ports
get_modules	report_attributes

Module Specification

A module object (module_object) is specified by a module name which is a string. For example: “modA”.

Instance

Description

An instance is a single instantiation of a module. No instance object exists until the set_current_design command is executed.

The instance object type interacts with attributes as follows:

- Module attributes are automatically visible as read-only attributes on corresponding instances if the instance object type does not have the same attribute registered.
- Attributes defined with the applies_to_child_instances option set to On automatically become visible to all child instances of the instance object.

Built-In Attributes

ignore_during_icl _extraction	When set to true, even if the instance has an associated ICL module during ICL extraction, the instance is ignored and not reported in the extracted ICL. This attribute is useful to allow extracting the ICL for the portion of the network that is correct while other instances still have issues to resolve. The generated ICL module will have the tessent_ignored_design_instances attribute defined referencing all design instances that were ignored.
in_graybox	Set to true when the instance is included in the graybox netlist. Set to false when the instance is excluded from the graybox netlist, which is the default for all the instances in the design. You can change the attribute value with the analyze_graybox or set_attribute_value command. For more information about this attribute, refer to the analyze_graybox command description.
is_valid	Boolean that specifies whether the object still exists. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_block_id	The leaf_name of instances located inside generate blocks have the generate block ids joined by periods prepended to their instantiation name. This attribute returns the block_id part of the leaf_name. It is null for instances that are not inside generate blocks.
leaf_name	String that specifies the name of the instance (value of the name attribute) without the hierarchical path.

leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a look up table with <parent_module>.<leaf_nane_hash>.<pin_leaf_name> entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage.
master_module_name	String that specifies the value of the master_name on the module object associated with the instance.
module_name	String that specifies the value of the name of the module object associated with the instance.
name	String that specifies the hierarchical name of the instance relative to the current design. The name of every instance is unique.
object_type	Returns the object data type “instance”.
parent_instance	String that specifies the instance name in which the instance exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
power_domain_name	Returns the name of the power domain associated to the instance. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins on the instance will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.

Inheritance

Built-in attribute inheritance from associated module objects:

- cell_type
- has_no_definition
- ijtag_is_icl_module
- is_created

- is_hard_module
- is_incomplete
- is_modified
- is_power_always_on_cell
- is_power_isolation_cell
- is_power_level_shifter_cell
- is_power_retention_cell
- is_unsaved
- language
- library_name
- preserve_boundary
- type
- vhdl_architecture

Related Commands

get_attribute_list	get_name_list
get_attribute_value_list	report_attributes
get_common_parent_instance	

Instance Specification

An instance object (instance_object) is specified by a complete hierarchical instance pathname. For example: “/u1/u2/u3”.

Port

Description

Ports have a simple non-hierarchical name. A port is an object on a module and can have an input, output or inout direction.

One port object exists for each bit of a bused port so every bit of a bussed port can be attributed independently.

The port object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types. These attributes are stored on the associated port of the pins and not on the pin object itself.

Built-In Attributes

bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.
constraint_value_ during_load_unload	String that specifies a constant value for a top-level port during load_unload test procedure retargeting. Possible values are: 0, 1, X, or Z. Constraining the value of the top-level port enables tracing of signals from the core level to the chip level.
direction	String that specifies the direction of the port. Possible values are: input, output, or inout.

function	String that either reports the function of the port on cell modules or reports the presence of the power or ground attributes on ports of the current_design. For cell modules, this attribute is initially null and is set by the set_current_design command from information extracted from the cell library files. For information about how cell_type information can be loaded, refer to the read_cell_library command. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. See *Tessent Cell Library Manual* for a list of available function values. For a top-level module, this attribute is initially null and you can set one of the following values on an input/inout port with the set_attribute_value command: • **power** — Constrains power input/inout ports during circuit setup and DRC. • **ground** — Constrains ground input/inout ports during circuit setup and DRC. For more information about using these values, refer to "Excluding Power and Ground Ports from the Pattern Set" in the *Tessent Scan and ATPG User's Manual*.
gate_pin_id	Returns the id of a gate_pin when the pin has an associated gate_pin inside the flat model. Exists for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the port is of direction output and the port net is undriven or tied.
ignore_for_graybox	Set to true when an output port is to be excluded from graybox analysis. By default, this attribute is set to false for all output ports (considered by the graybox analysis backtracing). Setting this attribute on an input port has no effect on the analyze_graybox command. You need to exclude the scan_out ports of the core chains (or channel outputs if it's a design containing EDT hardware) from graybox analysis using this attribute because the tool has no way of knowing which outputs are core chain scan output ports. For more information about this attribute, refer to the analyze_graybox command description.
ijtag_c0	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_low_output_port_list attribute on the ICL module.

ijtag_c1	Boolean attribute that is set on the design ports of a module that has a corresponding ICL module, if the port name is listed in the forced_high_output_port_list attribute on the ICL module.
ijtag_connection_rule_ option	An attribute that reflects the connection_rule_option attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_high_ output_port_list	An attribute that reflects the force_high_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_forced_low_ output_port_list	An attribute that reflects the force_low_output_port_list attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_function	String that reflects the port function of the ports as described in the ICL module. Its value, which is populated in the ICL extraction mode of the patterns -ijtag context, is the port function with an underscore separating the words. For example, a ScanInPort in ICL has icl_function equal to scan_in.
ijtag_function_modifier	An attribute that reflects the function_modifier attribute of the ICL port of the ICL module associated to the design module during ICL extraction.
ijtag_logical_hier_ connection_from_src	An attribute that is set on output ports and contains the port on the module that is the logical source of the output port. It is extracted from the associated ICL module attribute ijtag_logical_source = “{port1 port2} {port3 port4} ...”
ijtag_logical_hier_ connection_to_dst	This attribute is set on input ports and contains the ports on the module that are the logical destinations of the input port. It is extracted from the associated ICL module attribute ijtag_logical_source = “{port1 port2} {port3 port4} ...”
index	Integer that specifies the index of the port that is already part of the name when the port is part of a bus.
is_bus	Boolean that specifies whether the port is part of a bus. This attribute is set to true when the port is part of a bus.
is_excluded_from_ isolation_constraints	Specifies that during retargetable pattern generation, the input ports with this attribute should not be constrained to X. This attribute is only used during core-level pattern generation (“set_current_mode -type internal” and patterns -scan context).
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
library_name	String that specifies the library the port belongs to.

master_module_name	String that specifies the value of the master_name on the module object the port belongs to.
module_name	String that specifies the name of the module object the port belongs to.
name	String that specifies the name of the port object. One port object exists for each bit of a bused port and the name includes the index of the port. For example: A[3]
object_type	Returns the object data type “port”.
power_domain_name	Returns the name of the power domain of the current design. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, all pins of the port will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the module. Any attribute stored on the hierarchical pin will be visible in the flat model. If you set this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
special_drive	String value extracted from the cell library data. See the read_cell_library command for information on how cell_type information can be loaded. Note that the LogicVision scang.lib, pad.lib and cell.lib format are natively supported by the read_cell_library command in case your cell library file only includes the ATPG cell modeling information. Possible values are: open_drain | open_source | pull0 | pull1.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, Z, or “” (empty string). The default is the empty string which means the port is not tied to a value.

Inheritance

Built-in attribute inheritance from associated objects:

- None

Related Commands

get_attribute_list	get_common_parent_instance
get_attribute_value_list	get_name_list
get_fanins	report_attributes
get_fanouts	

Port Specification

A port object (port_object) is an input, output, or inout interface of a module, which is specified by a string. Examples of port names can be “In1”, “ClkA”, and “Out3”.

Pin

Description

A pin is an object on an instance that can be connected to other pins or ports with net objects. No pin object exists until the set_current_design command is executed.

The pin object type interacts with attributes as follows:

- Attributes defined on ports are automatically visible as read-only attributes on the associated pin objects unless the attribute is also registered for the pin object type. This is the case for the name attribute which is defined for both port and pin object types.
- Port attributes are stored on the associated port of the pin and not on the pin object itself.
- Attributes defined on a net are automatically visible by a pin having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This behavior enables you to register an attribute for net object types and have the attribute fully visible from the root-level ports and the hierarchical pins.

Built-In Attributes

base_name	String that specifies the name of the pin without the leaf name.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_pin_id	Returns a gate_pin id when the pin has an associated gate_pin inside the flat model. Set to true for cell pins associated to fault sites and for hierarchical pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.
has_functional_source	Returns true unless the pin is of direction input and the pin is not connected, is connected to an undriven or tied net or is connected to a constant.
ijtag_logical_hier_ connection_from_src	Reflects the result of the add_ijtag_logical_source commands. Each destination points back to its logical source.
ijtag_logical_hier_ connection_to_dst	Reflects the result of the add_ijtag_logical_source commands. Each source points forward to its logical destinations.
is_hard_module	Boolean that reports the state of the is_hard_module attribute on the instance on which the pin exists.
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.

leaf_name	String that specifies the name of the pin (value of the name attribute) without the hierarchical path.
name	String that specifies the hierarchical name of the pin object relative to the current design. One pin object exists for each bit of a bused port; the name includes the index of the port appended to its associated instance name. For example: u1/u2/A[3].
object_type	Returns the object data type “pin”.
parent_instance	String that specifies the instance name of the instance on which the pin is instantiated. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the <leaf_instance_name>/<leaf_pin_name> part of the path. For example: parent_instance of pin (u2/u7/a) is u2 parent_instance of pin (u2/b) is null
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute on the pin’s parent instance is set to true. This attribute is set to true if the is_hard_module attribute on the pin’s parent instance is set to true. The default is false.
power_domain_name	Returns the name of the power domain of the instance to which this pin belongs. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.
preserve_boundary	Boolean that specifies whether to preserve the boundary in the flat model. When set to true, the hierarchical pin will have an inferred buffer created in the flat model to preserve the separation of nets inside and outside the instance. Any attribute stored on the hierarchical pin will be visible in the flat model. If you enable this attribute on a pin of the design, the flat model will need to be reconstructed. The attribute is not lost during flattening.
tie_value	String that indicates the value to which the pin is tied. Possible values are: 0, 1, X, Z, or “” (empty string). The default is the empty string which means the pin is not tied to a value.

Inheritance

Built-in attribute inheritance from associated port objects:

- bus_left_index
- bus_name
- bus_range_direction
- bus_right_index
- bus_width
- direction

- function
- ijtag_function
- index
- is_bus
- library_name
- master_module_name
- module_name
- preserve_boundary
- special_drive

Related Commands

get_attribute_list	get_name_list
get_attribute_value_list	get_pins
get_fanins	report_attributes
get_fanouts	

Pin Specification

A pin object (pin_object) is specified by a complete hierarchical pin pathname including the pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Net

Description

No net objects exist until the set_current_design command is executed.

For every port and pin object, a net object with the same name exists. When objects are referenced by name in the editing and attribute commands, if a specified name exists as both a pin and a net or a port and a net, the pin or port is selected. If you want to force the selection of the net object that has the same name as a pin or port, use the get_nets introspection command. For examples of this usage, see the examples shown on the move_connections command reference page.

The net object type interacts with attributes as follows:

- Attributes defined on a net are automatically visible by a pin or port having the exact same name unless the attribute is also defined on the pin object type. The reverse is also true. Attributes defined on a pin or port are automatically visible by a net having the exact same name unless the attribute is also defined on the net object type. This is true for attributes on pins inherited from their associated port. This behavior enables you to register an attribute for the net object type and have the attribute fully visible from the hierarchical pins and from the ports of the current design.

Built-In Attributes

bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction. This attribute follows the to and downto BIT_VECTOR types in VHDL.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.
has_functional_source	Returns true unless the net is undriven or tied.
has_rtl	Boolean that indicates whether the net is connected to an RTL construct, such as an always block. Also indicates when the driver of the net is an RTL construct. The attribute is true under the following circumstances: • The net is used in an RTL expression. • The net is directly connected to a complex pin (not via an assign). The attribute is always false in the -no_rtl flow.

index	Integer that specifies the index of the net that is already part of the name when the net is part of a bus.
in_graybox	Set to true when the net is included in the graybox netlist. Set to false when the net is excluded from the graybox netlist, which is the default for all the nets in the design. The attribute value can be changed by the analyze_graybox or set_attribute_value command. For more information about this attribute, refer to the analyze_graybox command description.
is_bus	Boolean that specifies whether the net is a part of a bus. This attribute is set to true when the net is part of a bus.
is_valid	Boolean that specifies whether the object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_block_id	The leaf_name of nets located inside generate blocks have the generate block ids joined by periods prepended to their local name. This attribute returns the block_id part of the leaf_name. It is null for nets that are not inside generate blocks.
leaf_name	String that specifies the name of the net (value of the name attribute) without the hierarchical path.
leaf_name_hash	This attribute is normally identical to the leaf_name attribute except when the leaf instance is located inside a generate loop construct inside an RTL file. The leaf_name includes the block id with the loop index such as in loopA[4].u2. The leaf_name_hash attribute is the same except that the loop index is replaced by a hash sign as shown here: loopA[#].u2. You use this attribute if you are editing the content of generate loops and you have not enabled auto uniquification. For the delete_connections and move_connections commands, you must use a lookup table with <parent_module>.<leaf_nane_hash>.<pin_leaf_name> entries to know if you have already processed that pin. See the examples in the move_connections and get_dft_cell command descriptions for examples of such usage.
library_name	String that specifies the library of the module in which the leaf net resides.
master_module_name	String that specifies the value of the master_name in which the leaf net resides.
module_name	String that specifies the name of the module object in which the leaf net resides.
name	String that specifies the hierarchical name of the net object relative to current design. One net object exists for each bit of a bused net; the name includes the index of the net appended to its associated instance name. For example: u1/u2/A[3].

object_type	Returns the object data type “net”.
parent_instance	String that specifies the instance name in which the net exists. This attribute automatically handles the escaped identifier and returns the instance path of the instance without the /<leaf_name> part of the path.
parent_is_hard_module	Boolean that specifies whether the module containing the net has the is_hard_module attribute set to true. This attribute is set to true if the module containing the net has the is_hard_module attribute set to true. The default is false.
power_domain_name	Returns the name of the power domain of the instance in which this net exists. This attribute is populated when going from setup to analysis system mode if a UPF or CPF file has been read in.
tie_value	String that indicates the value to which the net is tied. Possible values are: 0, 1, X, Z, or “” (empty string). The default is the empty string which means the net is not tied to a value.

Inheritance

Built-in attribute inheritance from associated pin objects:

- Attributes defined on pins or ports having the same name as the net, if the attribute is not also registered on the net object type.

Related Commands

get_attribute_list	get_name_list
get_attribute_value_list	get_nets
get_fanins	report_attributes
get_fanouts	

Net Specification

A net object (net_object), sometimes called just a net, is a wire that connects the pins of instances in the design. A net is specified by a hierarchical net pathname. In the following example, u5/u2/Q is a pin that drives the net u5/u2_Q. This example is based on the example design presented in the “Hierarchical Design Data Model” section of the *Tessent Shell User’s Manual*.

SETUP> get_fanouts u5/u2/Q

```
{u5/u1/A0 u5/u3/A0}
```

SETUP> get_fanouts u5/u2/Q -stop_on net

```
{u5/u2_Q}
```

Pseudo_port

Description

A pseudo_port represents a user-added primary input or primary output. A pseudo_port object enables the introspection of user-added primary inputs and outputs such as those created using the commands “add_primary_inputs”, “add_primary_outputs”, “add_primary_inputs -internal”, “add_primary_outputs -internal”, and “add_clocks -internal”.

A pseudo_port is not a real port in that it is not present in the original design, it cannot be used for design editing, and it is not reflected in designs written out using the write_design command. However, the pseudo_port is reflected in the flat model and it is used for analysis operations that use the flat model such as ATPG and simulation.

A pseudo_port shares some of the same attributes as the port object. Pseudo ports have a simple non-hierarchical name which may be the same as the pin's hierarchical name (for non-grouped pseudo_ports).

Pseudo_ports can be introspected with the get_ports command. The “real” pins that underlie a pseudo_port can be introspected with the “get_pins -of_pseudo_ports *pseudo_port_objects*” command.

The pseudo_port object type does not support attribute inheritance to the real underlying objects (pins/nets) associated to it. That is, the real design object behind the pseudo_port does not see the attributes set on this type of port.

Built-In Attributes

constraint_value_ during_load_unload	String that specifies a constant value for a top-level pseudo_port during load_unload test procedure retargeting. Possible values are: 0, 1, X, and Z. Constraining the value of the top-level pseudo_port enables tracing of signals from the core level to the chip level.
direction	String that specifies the direction of the pseudo_port. Possible values are: input and output.
gate_pin_id	Returns the id of the gate_pin associated with the pseudo_port inside the flat model.
is_bus	Boolean that specifies whether the pseudo_port is part of a bus. This attribute is always set to false. The pseudo_port object type does not support busses.
is_valid	Boolean that specifies whether the object still exists.
name	String that specifies the name of the pseudo_port object.
object_type	Returns the object data type “pseudo_port”.

Inheritance

Built-in attribute inheritance from associated objects:

- None

Related Commands

add_clocks	get_fanins
add_primary_inputs	get_fanouts
add_primary_outputs	get_name_list
get_attribute_list	get_ports
get_attribute_value_list	report_attributes

Flat Design Data Model

The flat design data model is an internal, flattened representation of the hierarchical design that is produced when exiting setup mode. You can also explicitly create the flat model using the create_flat_model command.

The flat design data model consists of gates that are connected together. A gate is an instance of a primitive module, and a gate_pin object represents a pin on a gate instance. Gates and gate_pins have no unique instance name, instead having a unique ID that is available to differentiate one from another. The format of the ID is two integers separated by a period character. The first integer represents the gate and the second integer represents the gate_pin. However, this ID does not remain in place from one netlist version to the next or from one Tessent Shell invocation to the next. For that reason, you should avoid hardcoding a gate ID into a script.

Flat Design Object Types

The gate_pin is the only object type in the flat model.

Gate_pin

Description

A gate_pin represents a pin on a gate instance in a flattened design.

Built-In Attributes

The gate_pin object type has the following attributes:

base_name	String that specifies the name of the instance in which the primitive module of the gate_pin is instantiated.
direction	String that specifies the direction of the gate_pin object: input (default) or output.
fault_site	String that is populated in the analysis mode of the patterns -scan context and reports if the pin is a fault site for 0, 1, or both fault polarity types. The attribute is null on pins that are not a fault site for both polarities.
gate_id	Integer that specifies the unique ID of the gate to which the gate_pin belongs. This integer is identical to the first integer in the gate_pin's id attribute. This value is not persistent and changes every time you create a flattened model.
hier_pin_name	String that specifies a hierarchical pin name when the gate_pin is associated to a hierarchical pin. This attribute is null when the gate pin is not associated to a hierarchical pin. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.
id	Specifies a unique ID for each gate_pin that consists of two integers separated by a period: *gate_id.pin_index*
is_valid	Boolean that specifies whether the gate_pin object still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the hierarchical pin name without the hierarchical path when the gate_pin is associated to a hierarchical pin. This is null by default.
name	String that specifies name of the gate_pin object. The value is the same as the hier_pin_name attribute when not empty; otherwise, the value is the same as the id attribute of the gate_pin object. Pins on hierarchical instances of type cells and pins on design modules that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true are the only hierarchical pins with a corresponding gate_pin.

object_type	Returns the object data type “gate_pin”.
parent_instance	If the gate_pin is associated to a hierarchical pin, this string specifies the name of the instance on which the hierarchical pin is instantiated. If the gate_pin isn't associated to any hierarchical pin, this attribute specifies the name of the instance on which the primitive module is instantiated.
parent_is_hard_module	Boolean that specifies whether the is_hard_module attribute is set to true on the parent module of the gate_pin. This attribute is false if the parent module of the gate_pin does not have the is_hard_module attribute set to true. This attribute is true for primitives instantiated directly into a design module which does not have is_hard_module set to true. This attribute can also be true for a gate_pin that has the hier_pin_name pointing to a pin on a module instantiated into a non-hard design module.
pin_index	Integer that specifies the *id* of the pin. This integer is identical to the second integer of the gate_pin’s *id* attribute. The pin index is persistent. The pin_index is the index of the gate_pin on the primitive module, where 0 is the index of the single output and integers larger than 0 correspond to the inputs.
primitive_name	Name of the primitive module associated with the gate on which the gate_pin exists. The primitive_name attribute values for all primitive types are listed in Table 3-1.
primitive_port_name	Name of the port on the primitive module. The primitive_port_name attribute values for all primitive types are listed in Table 3-1.
power_domain_name	Returns the name of the power domain in which the gate (instance) to which this gate_pin is attached exists. This attribute is inherited from the instance.
simulation_value	String that contains the simulation value on a gate_pin.

Built-In Primitives and Primitive Port Names

Table 3-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	**Outputs**
BUF	Buffer	I	O
INV	Inverter	I	O
AND	AND gate	I0, I1, …, In	O
NAND	Negated AND	I0, I1, …, In	O
OR	OR gate	I0, I1, …, In	O

Table 3-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	**Outputs**
NOR	Negated OR gate	I0, I1, …, In	O
XOR	Exclusive OR	I0, I1, …, In	O
NXOR	Negated Exclusive OR	I0, I1, …, In	O
DFF	D Flip-flop	S, R, CK (CK0, CK1, … CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
DLAT	D latch	S, R, CK (CK0, CK1, … CKn in case of multiple clock/data input pairs), D (D0, D1, ..., Dn in case of multiple clock/data input pairs)	Q
PI	Primary input	-	O
PO	Primary output	I	-
TIE0	Tie-0	I	O
TIE1	Tie-1	I	O
TIEX	Tie-X	I	O
TSD	Tri-state driver with active high at control input.	EN, D	O
BUS	Bus primitive. Drivers must include at least one TSH or switch gate.	I0, I1, …, In	O
ZVAL	Z converter gate, converts Z to X: dummy gate inserted between output of the gate taking Z and the binary gate input.	I	O
WIRE	Wire	I0, I1, …, In	O

Table 3-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	**Outputs**
MUX	Multiplexer	S, D0, D1	O
TIEZ	Tie-Z	I0, I1, …, In	O
NMOS	Switch gate	G (gate), S (source)	D (drain)
ZHOLD	Z-hold allows modeling of tri-state nets such that they can retain the previous state when the net goes to a Z value.	I	O
PBUS	Pull-bus gate. Used when combining strong (ZW) bus and weak bus signals (e.g. TIE0) together. The strong value (input 1) always goes to the output, unless the value is Z, in which case the weak value (input 2) propagates to the output.	I0, I1	O
WBUS	Weak bus primitive	I0, I1, …, In	O
BUFZ	Buffer allowing Z value at input.	I	O
XDET	X-detector gate: input ‘X’ results in output value ‘1’; otherwise output is ‘0’.	I	O

Table 3-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	**Outputs**
ZDET	Z-detector gate: input ‘Z’ results in output value ‘1’; otherwise output is ‘0’.	I	O
RAM	Read-write memory	S, R, WCK (WCK0, WCK1, … WCKw in case of multiple write ports), WEN (WEN0, WEN1, … WENw in case of multiple write ports), WADDR[0…i] (WADDR0[0…i], WADDR1[0…i], … WADDRw[0…i] in case of multiple write ports), WDIN[0…j] (WDIN0[0…j], WDIN1[0…j], … WDIN w[0…j] in case of multiple write ports), RCK, REN, RADDR[0…i] (RADDR0[0…i], RADDR1[0…i], … RADDRr[0…i] in case of multiple read ports)	Q
ROM	Read-only memory	CK, EN, ADDR[0…i] (ADDR0[0…i], ADDR1[0…i], … ADDRr[0…i] in case of multiple read ports)	Q
RAM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, …, In	O
ROM_OUT	Output of a memory, place holder for multiple data outputs of memories in the flat model.	I0, I1, …, In	O

Table 3-1. Built-in Primitives and Primitive Port Names

Primitive Name	Functional Description	Primitive Port Names	
		Inputs	Outputs
FB_BUF	Feedback buffer: internal gate type to break combinational loops in the design.	I	O
PGEN	Pulse generator: circuitry that creates a pulse at its output when active.	I	O
CORE	Core (obsolete)	I0, I1, …, In	O
CORE_OUT	Output of core, place holder for multiple core outputs in the flat model (obsolete).	I0, I1, …, In	O

Inheritance

Built-in inherited attributes:

- During flattening, attributes defined on hierarchical pins (cell or design) are copied to the gate_pin objects when there is a gate_pin associated to the hierarchical pin. Those are fault sites on cell pins, and those pins that have an attribute that is set to a non-default value and whose -preserve_boundary_in_flat_model option is set to true.

Related Commands

get_attribute_list	get_gate_pins
get_attribute_value_list	get_pins
report_attributes	get_name_list

Gate_pin Specification

A gate_pin object (gate_pin_object) is assigned a number ID rather than having an actual name. The number ID is a unique gate pin object id of the format "gate_id.gate_pin_id". For example an output of an AND gate in the flattened design might have an ID of “1325.0”.

ICL Data Model

The purpose of the Instrument Connectivity Language (ICL) is to describe the elements that comprise the 1687 network as well as their logical (though not necessarily physical) connections to each other and to the instruments at the endpoints of the network. ICL bears a loose resemblance to a hierarchical netlist such as Verilog; it is organized by modules that may contain instances of other modules, and it describes the connections between the pins of the instances. However, it is important to note that ICL is not a complete netlist; connections are port-to-port rather than through nets, and ICL freely uses abstraction in order to omit the detailed physical construction of the circuitry — only the behavioral operation of the network is represented.

Beyond merely documenting the structure of the network, the purpose of ICL is to enable an operation called “test pattern retargeting.” With test pattern retargeting, the sequences of test patterns, written in PDL, that are to be applied at the instrument boundaries are mapped through the IJTAG network to a higher level of hierarchy and ultimately to chip level pins. Tessent Shell reads and stores ICL information in a data structure and uses that information to retarget the instrument-level test patterns into test patterns that can then be applied at a higher level.

The ICL data model defines objects as one of the following four data types: icl_module, icl_instance, icl_port, and icl_pin. The icl_module and the icl_port object types are created as soon as ICL modules are read in using the read_icl command. Each icl_module object has its list of icl_port objects. Once the current design is set using the set_current_design command, the ICL data model is elaborated downward from the current design and icl_instance objects are created for each instance of icl_module, and icl_pin objects are created for each port of the modules associated to the icl_instances. The icl_instance and icl_pin objects are hierarchical objects and therefore only exist after the current design is set.

If a Verilog or Vhdl representation has also been read prior to issuing the set_current_design command, the port list on the design and the ICL module corresponding to the current design is compared to make sure that all ports declared in the ICL module also exist on the design module. The design module is allowed to have more ports that the ICL module. In fact, it is very likely the design module will have more ports because the ICL module only include the ports with an IJTAG function while the design module includes all ports on the physical module. The test bench created for the current design uses the complete port list read in from the design view when it is supplied.

ICL Object Types

The icl_module, icl_instance, icl_port, and icl_pin object types are described on the following reference pages.

icl_module

Description

One module is designated as the target module using the set_current_design command. Instances and pins are hierarchical objects defined relative to the current module.

The following properties apply to the icl_module object type:

- An icl_module object has a simple non-hierarchical name. Only one icl_module with a given name is allowed within a given namespace. Currently there is only one global namespace for ICL in Tessent Shell.
- An icl_module can be parameterized and its parameters can be overridden during instantiation. For each combination of parameter overrides, an icl_module object corresponding to the parameterized view exists. The name of the parameterized view is uniquified with a @PV# suffix where # is a unique integer starting from 1 and incrementing per unique parameterized view.
- One master icl_module exists for each module definition in the input files. The master icl_module is the one that contains the ICL module definition with default parameter values.
- Attributes defined on icl_modules are automatically visible as read-only attributes on the instance objects when the same attribute name is not also registered on the icl_instance object type.
- Attributes defined with the set_attribute_options -applies_to_child_instances switch set to On automatically become visible to all child instances of each instance of the icl_module.
- icl_module attributes are stored on all parameterized views of an icl_module. When an attribute value is set on an icl_module object, the attribute value is set on all icl_module objects having the same master_name within a name space.

For more information on attributes, see “Attributes” on page 103.

Built-In Attributes

The icl_module object type has the following attributes:

file_path_name	String that specifies the pathname to the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been saved and read back in.
forced_high_input_port_list	Lists all input ports that were constrained to 1 during ICL extraction. Those constraints are automatically re-applied in the generated patterns. They are also verified to be properly simulated when running ICL extract at the next level. An I5 DRC violation is generated if the simulated value on any of those pins is not 1.

forced_high_internal_ input_port_list	Lists all internal pins that were constrained to 1 during ICL extraction. Those constraints are reported in this attribute for debugging and serve no other purpose.
forced_high_output_port_ list	Lists output ports of IJTAG instruments that must be forced high during ICL extraction in order to see all the connections between the instruments. You must use this attribute with care and typically use it for logictest testmode signals sourced by the ICL network itself. For example, if you have a TDR bit driving a TM signal and this TM signal puts the circuit in scan mode, you may need to force this signal to its inactive value for the rest of the ICL network to be traceable.
forced_low_input_port_ list	Lists all input ports that were constrained to 0 during ICL extraction. Those constraints are automatically re-applied in the generated patterns. They are also verified to be properly simulated when running ICL extract at the next level. An I5 DRC violation is generated if the simulated value on any of those pins is not 0.
forced_low_internal_ input_port_list	Lists all internal pins that were constrained to 0 during ICL extraction. Those constraints are reported in this attribute for debugging and serve no other purpose.
forced_low_output_port_ list	List output ports of IJTAG instruments that must be forced low during ICL extraction in order to see all the connections between the instruments. You must use this attribute with care and typically use it for logictest testmode signals sourced by the ICL network itself. For example, if you have a TDR bit driving a TM signal and this TM signals puts the circuit in scan mode, you may need to force this signal to its inactive value for the rest of the ICL network to be traceable.
icl_extraction_date	Reports the date and time when the ICL module was extracted. It is defined in the ICL module and created by ICL extraction.

ijtag_logical_connection	An attribute that creates a logical connection between source and destination ports. These attributes are transferred to the ijtag_logical_hier_connection_from_src and ijtag_logical_hier_connection_to_dst attributes on the matching module object. ICL extraction traces through those connections as if there was a buffer between the two ports. The syntax is attribute ijtag_logical_connection = "{src1 dst1} {src2 dst2} ..."; The source and destination ports may or may not be declared as ports in the ICL module but they must be declared with the proper direction in the design module object. To create a logical connection that is not between an input and output port of a module having an associated ICL module, use the add_ijtag_logical_connection command.
is_created	Boolean that specifies whether the module was created by ICL extraction. This attribute is set to true for modules created by ICL extraction. This attribute is not exported to the ICL file. Once the ICL module is re-read, this attribute is false.
is_modified	Boolean that specifies whether an ICL module contains attributes that have been added or modified after the module was read in.
is_valid	Boolean that specifies whether the ICL module is deleted or still valid. This Boolean attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
keep_active_during_scan_test	Boolean that specifies that the given ICL module is a node that must remain active during scan test so as not to be disturbed and be usable for scan test_setup. See the create_dft_specification command description for more information on how it is used.
line_number	Integer that specifies a line number in the file where a given module was read from. This attribute is null for modules created by ICL extract until they have been saved and read back in.
master_name	String that specifies the name of the icl_module without the optional parameterized view suffix.
name	String that specifies the name of the icl_module with an optional parameterized view suffix @PV#. ICL modules are defined from files read into the ICl database with the read_icl command. For example, the name of a parameterized view of the master icl_module ModA is: ModA@PV1
object_type	Returns the object data type “icl_module”.

partially_forced_input_port_list	Lists all bus ports with at least one element in the forced_low_input_port_list and forced_high_input_port_list. This attribute is useful when creating the testbench using the ICL port list in order to get the complete bus range of the constrained bused ports. This attribute is not used when the Verilog or VHDL design module is read before setting the current design in the patterns -ijtag context because the top-level port list is then extracted from the design module.
scan_interface_list	Lists all ScanInterface names defined within the ICL module.
tessent_dft_function	A string attribute used to identify an ICL module having special meaning during DFT. For example, the create_dft_specification command uses this attribute to identify the Sib(sti) and Sib(sri) so that they can be found again during the next insertion pass.
tessent_ignored_design_instances	A string attribute reported by ICL extract to reflect the list of design instances ignored during ICL extraction. These are the design instances with an ICL module for which the Boolean attribute ignore_during_icl_extraction has been set. The format of this attribute is a list of three-element lists as shown here: { {<IclModuleName>} {<DesignModuleName>} {<DesignInstancePath>} ... }
tessent_instrument_container	String that specifies the base name of the directory that contains the Tessent Instrument Description (TID) file and any other IP-specific data. This attribute is required for any ICL module that has IP-specific data.
tessent_instrument_type	String that identifies the ICL module as one of the known instrument types within Tessent Shell. This attribute is currently unused and is a provision for the open plug-in architecture being developed.
tessent_is_physical_module	Boolean that is set on the top ICL module when the extract_icl command runs and the level was set to chip or physical_block using the set_design_level command,
tessent_rtl_pre_trace_stop_port_list	Lists the names of ports on which the tool stops the ICL tracing during the ICL extraction pre-trace phase. This is useful to prevent the unnecessary synthesis of module during ICL extraction.
tessent_signature	A string that is used to identify that an ICL module as not been modified after its creation. An ICL module can be read in without a tessent_signature or with a valid tessent_signature but a module with an invalid signature generates an errors and is rejected during the read_icl command.

test_setup_procfile	Reports the filename of the procfile that contained the optional test_setup procedure used during ICL extraction. It currently is only informational to remind the user to reuse the same test_setup file during IJTAG pattern retargeting.
tessent_use_in_dft_ specification	String that you can set on an ICL module in its ICL file to force the create_dft_specification command to ignore instances of this module. The legal value is auto or false.

Related Commands

get_attribute_list	get_icl_scan_interface_port_list
get_attribute_value_list	get_iproc_argument_default
get_icl_fanins	get_iproc_argument_list
get_icl_fanouts	get_iproc_body
get_icl_instances	get_iproc_list
get_icl_modules	get_name_list
get_icl_module_parameter_list	report_attributes
get_icl_module_parameter_value	report_icl_modules
get_icl_ports	report_module_matching
get_icl_scan_interface_list	report_module_matching_options

ICL Module Specification

An ICL module object is specified by an ICL module name which is a string. For example: “modA”.

icl_port

Description

The following properties apply to the icl_port object type:

- An icl_port object has a simple non-hierarchical name.
- An icl_ port is an object on an icl_ module and can have an input or output direction.
- One port object exists for each bit of a bused port so every bit of a bussed port can be attributed independently.
- Attributes defined on icl_port objects are automatically visible as read-only attributes on the associated icl_pin objects unless the attribute is also registered for the icl_pin object type.
- Attributes set on an icl_port of one icl_module are set on the same port of each icl_module having a common master_name. Because different parameterized views can have more or less icl_port objects when the size of a bussed icl_port is parameterized, attributes on the other icl_module having the common master_name are only copies if the port object actually exists on the given parameterized view.

For more information on attributes, see “Attributes” on page 103.

Built-In Attributes

The icl_port object type has the following attributes

active_polarity	String attribute that is defined on the port with the function Reset to reflect the ActivePolarity of the ResetPort in ICL.
bus_left_index	Integer that specifies the left index of the complete bus. Null for scalar ports.
bus_name	String that specifies the name of the bus without the index. <name> = bus_name [index] when is_bus is true; otherwise, bus_name is null.
bus_range_direction	String set to “up” or “down” that reflects the bus range direction.
bus_right_index	Integer that specifies the right index of the complete bus. Null for scalar ports.
bus_width	Integer that specifies the number of bits independent of whether left_index is smaller, larger than, or equal to right_index.

connection_rule_option	Specified inside the Port wrappers of the ICL modules and used to modify the connection rules for the corresponding port. You use one of the following values to achieve the described behavior. • **allowed_no_source** — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • **allowed_tied** — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high or low. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • **allowed_tied_low** — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied low. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • **allowed_tied_high** — Allowed on input ports of an ICL instrument to indicate that this input port is allowed not to be connected to a valid source as long as the port is tied high. You typically use this for DataInPort and ScanInPorts of host ports which are not always used in all instantiations. • **allowed_no_destination** — Allowed on output ports of an ICL instrument to indicate that this output port is allowed not to be connected to a valid destination. You typically use this for DataOutPort and ScanOutPorts of host ports which are not always used in all instantiations. • **auxiliary_data_enable** — Identifies the DataOutPort as an auxiliary data enable pin. When tracing a port with a forced_low_output_port_list or a forced_high_output_port_list attribute pointing to a port declared as an auxiliary_data_enable or auxiliary_data_enable_inverse, ICL extract asserts all ports on all instances declared as an auxiliary_data_enable or auxiliary_data_enable_inverse to their inactive values, except for the ones referenced by the current port which are forces to their active values instead. This enables ICL extract to extract DataOut ports connected through an AND OR mux like this: (d1 & en1) | (d2 & en2) | (d3 & en3);

connection_rule_option (continued...)	• **auxiliary_data_enable_inverse** — Identifies the DataOutPort as an auxiliary data enable inverse pin. • **must_connect_to_top_port** — Allowed on input and output ports of an ICL instrument to indicate that this input or output port is only allowed to be connected to a primary input or output pin of the current design. For example, a ScanInPort is allowed to be sourced by a ScanOutPort of another instance. If this attribute is set to must_connect_to_top_port, it is only allowed to be sourced by a primary input port.
default_load_value	This attribute corresponds to the DefaultLoadValue that is specified on DataIn ports in the ICL. Possible values are: 0, 1, or X, where X is specified when the DataIn port does not have a specified default load value.
direction	Direction of the icl_port. Possible values are: input or output.
forced_low_output_port_ list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force low during the tracing of the current output port. The listed ports are simulated low and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.
forced_high_output_port_ list	This attribute is specified inside the Port wrappers of the ICL modules and is used to list output ports on the IJTAG instrument that must be force high during the tracing of the current output port. The listed ports are simulated high and appended to the stable_after_setup simulation condition when tracing the associated output port. For example, tdoEnable typically needs to be forced active when tracing the tdo port of a TAP controller.
freq_multiplier	A positive integer value attribute at the icl_port. The value is computed from the optional FreqMultiplier keyword of the ICL ToClockPort statement. The default value is 0.
freq_divider	A positive integer value attribute at the icl_port. The value is computed from the optional FreqDivider keyword of the ICL ToClockPort statement. The default value is 0.

function_modifier	A string with value CaptureShiftClock or CaptureShiftClockInv that can be specified on a ToCaptureEnPort and a CaptureEnPort in ICL. When specified to CaptureShiftClock or CaptureShiftClockInv on a CaptureEnPort, the port can only trace to ToCaptureEnablePort or a CaptureEnPort with function_modifier also set to CaptureShiftClock or CaptureShiftClockInv. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClock, the port is driven by a gated version of TCK that is only active during the capture and shift cycles. When generating patterns for an ICL module and this module has a CaptureEnPort with function_modifier equal to CaptureShiftClockInv, the port is driven by a gated version of inverted TCK that is only active during the capture and shift cycles.
ijtag_function	String that specifies the function of the icl_port; for example, “scan_in”. This is the ICL port function string converted to lowercase and the words connected with underscores.
index	Integer that specifies the index of the icl_port that is already part of the name when the icl_port is part of a bus.
is_bus	Boolean that specifies whether the icl_port is part of a bus. This attribute is set to true when the icl_port is part of a bus.
is_valid	Boolean that specifies whether the ICL port still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
launch_edge	String attribute that is defined on a port with function ScanOut to reflect the LaunchEdge of the ScanOutPort in ICL.
master_module_name	String that specifies the master_name of the icl_module object the icl_port belongs to.
module_name	String that specifies the name of the icl_module object the icl_port belongs to.
name	String that specifies the name of the icl_port including its bus index when it is an element of a bus.
object_type	Returns the object data type “icl_port”.
scan_interface_list	Lists all ScanInterface names that the port belongs to.
tessent_dft_function	A string attribute used to identify ICL ports having special meaning during DFT.
tessent_use_in_dft_ specification	String that you can set on an ICL Port in its ICL file to force the create_dft_specification command to ignore this DataIn or DataOut port. The legal value is auto or false.

Inheritance

Built-in attribute inheritance from associated icl_module objects:

- None

Related Commands

get_attribute_list	get_iclock_option
get_attribute_value_list	get_name_list
get_icl_fanins	report_attributes
get_icl_fanouts	report_iclock
get_icl_ports	report_ijtag_logical_connections
get_iclock_list	

Port Specification

An ICL port object is an input, output, or inout interface of an ICL module, which is specified by a string. Examples of ICL port names are: “In1”, “ClkA”, and “Out3”.

icl_instance

Description

An icl_instance is a single instantiation of an icl_module. No icl_instance object exists until the set_current_design command is executed.

The icl_instance object type interacts with attributes as follows:

- Module attributes are automatically visible as read-only attributes on corresponding icl_instances if the icl_instance object type does not have the same attribute registered.
- Attributes defined with the applies_to_child_instances option set to On automatically become visible to all child instances of the icl_instance object.

For more information on attributes, see “Attributes” on page 103.

Built-In Attributes

The icl_instance object type has the following attributes:

is_valid	Boolean that specifies whether the ICL instance still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the icl_instance (value of the name attribute) without the hierarchical path.
master_module_name	String that specifies the value of the master_name on the icl_module object associated with the icl_instance.
module_name	String that specifies the name of the ICL module associated to the icl_instance with the @PV# suffix when the instantiation uses parameter overrides.
name	String that specifies the full hierarchical instance path of the icl_instance with '.' as a hierarchy separator.
object_type	Returns the object data type “icl_instance”.
parent_instance	String that specifies the instance name of the ICL instance in which the pin is instantiated. It is null for pins on icl_instances found within the current design.
tessent_design_instance	An attribute populated by ICL extract when extracting the top level ICL module. The attribute reflects the design instance name corresponding to the ICL instance at the time of ICL extraction.

Inheritance

Built-in attribute inheritance from associated icl_module objects:

- c0_constrained_internal_pin_list

- c0_constrained_pin_list
- c1_constrained_internal_pin_list
- c1_constrained_pin_list
- icl_extraction_time
- is_created
- partially_constrained_bus_pin_list
- port_group_list
- tessent_instrument_type
- test_setup_procfile

Related Commands

get_attribute_list	get_icl_scope
get_attribute_value_list	get_name_list
get_icl_instances	report_attributes

ICL Instance Specification

An ICL instance object is specified by a complete hierarchical ICL instance pathname. For example: “/u1/u2/u3”.

icl_pin

Description

An icl_pin is an object on an icl_instance that can be connected to other icl_ pin or icl_port objects. No icl_pin object exists until the set_current_design command is executed.

The icl_pin object type interacts with attributes as follows:

- Attributes defined on icl_port objects are automatically visible as read-only attributes on the associated icl_pin objects unless the attribute is also registered for the icl_pin object type. This is the case for the name attribute which is defined for both icl_port and icl_pin object types.

For more information on attributes, see “Attributes” on page 103.

Built-In Attributes

The icl_pin object type has the following attributes

base_name	String that specifies the name of the pin without the leaf name.
freq_multiplier	A positive integer value attribute at the icl_pin. The value is computed from the optional FreqMultiplier keyword of the ICL ToClockPort statement. The default value is 0.
freq_divider	A positive integer value attribute at the icl_pin. The value is computed from the optional FreqDivider keyword of the ICL ToClockPort statement. The default value is 0.
is_valid	Boolean that specifies whether the icl_pin still exists. This attribute is set to true unless the object is deleted, in which case the is_valid attribute becomes false.
leaf_name	String that specifies the name of the ICL pin (value of the name attribute) without the hierarchical path.
name	String that specifies the full hierarchical name of the icl_pin with '.' as a hierarchy separator.
object_type	Returns the object data type “icl_pin”.
parent_instance	String that specifies the name of the parent icl_instance of the icl_instance on which the icl_pin is on.
tie_value	String that indicates the value to which the port is tied. Possible values are: 0, 1, X, or “” (empty string). The default is the empty string which means the port is not tied to a value.

Inheritance

Built-in attribute inheritance from associated icl_port objects:

- bus_left_index

- bus_name
- bus_range_direction
- bus_right_index
- bus_width
- direction
- ijtag_function
- index
- is_bus
- master_module_name
- module_name
- port_group_list

Related Commands

get_attribute_list	get_iclock_list
get_attribute_value_list	get_iclock_option
get_icl_fanins	get_name_list
get_icl_fanouts	report_attributes
get_icl_pins	report_ijtag_logical_connections
get_icl_instances	

Pin Specification

An ICL pin object is specified by a complete hierarchical ICL pin pathname including the ICL pin. For example: /u1/u2/u3/Y refers to pin Y of the hierarchical instance /u1/u2/u3.

Chapter 4
Attributes

Attributes are properties assigned to any object type. Attributes have names and values to associate relevant data with those objects. Attribute names may only include alphanumeric and underscore characters. This section provides information on the following topics:

Attribute-Related Commands 103
Attribute Types 104
Attribute Value Types 104
Attribute Filtering Equation Syntax 105
Glob and Regular Expression Pattern Matching Syntax 107
Matching Hierarchical Names With Escaped Identifiers 111

For detailed information on the attributes associated with each object type, see "Data Models" on page 57.

Attribute-Related Commands

You can use the set of Tessent Shell commands presented in Table 4-1 to create, manipulate, and query objects' attributes.

Table 4-1. Commands for Manipulating Attributes

Command	Description
get_attribute_list	Returns an alphabetically sorted Tcl list of attribute names.
get_attribute_option	Retrieves the current setting of an attribute's option.
get_attribute_value_list	Retrieves the value of an attribute on the specified design objects.
get_name_list	Returns the name(s) of the specified object(s).
register_attribute	Registers a new user-defined attribute.
report_attributes	Creates a human readable report for the set of attributes based on the usage options.
reset_attribute_value	Resets an attribute to its default value for the design objects specified in obj_spec.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in object_spec.

Table 4-1. Commands for Manipulating Attributes

Command	Description
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.

Attribute Types

Tessent Shell supports two types of attributes: pre-defined attributes and user-defined attributes. Different object types have different pre-defined attributes. You can also create and specify user-defined attributes.

Pre-Defined Attributes

Pre-defined attributes exist on the object itself in the internal database. Because of this, pre-defined attributes do not need to be registered. Different object types have different pre-defined attributes. By default, these attributes are read-only.

Each data model has a list of pre-defined attributes which are described in sections "Hierarchical Design Data Model", "Flat Design Data Model", and "ICL Data Model". All data models will be exposed in subsequent releases of Tessent Shell and their associated built-in attributes will be documented.

User-Defined Attributes

User-defined attributes are registered with the register_attribute command before they can be used. The registration operation defines the attribute's value type and its default value and determines which design object type the attribute belongs to.

Attribute Value Types

Attribute values can have the following types:

Table 4-2. Attribute Value Types

Type	Description
Boolean	Case-insensitive. A Boolean. • Accepts any of the following for true: 1, true, yes, on. • Accepts any of the following for false: 0, false, no, off
Integer	Case-insensitive. An integer.
Double	Case-insensitive. Double-precision floating-point number.
String	Case-insensitive. A sequence of alphanumeric characters.

Attribute Filtering Equation Syntax

An attribute filtering equation is a series of relations joined together with the && (AND) and || (OR) operators. Parentheses are also supported. A basic relation compares or matches the value of an attribute with a constant expression through a relational operator.

Relational Operators

The relational operators are:

==	Equal
!=	Not equal
>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
=~	Matches pattern
!~	Does not match pattern

Basic Relational Rules

The basic relational rules are as follows:

- Compare string attributes using any operator.
- Do not compare numeric attributes using pattern matching operators (=~ or !~).
- Compare Boolean attributes using only == and !=. The only possible value is true or false.
- Using an attribute without a relational operator in an equation is equivalent to saying att == true for a Boolean attribute or att != <defaultValue> for other types. You can also invert this default behavior by putting a ! in front of the attribute name. For example:

  ```
  is_something && !is_something_else
  ```

  translates to:

  ```
  is_something != <default> && is_something_else == <default>
  ```

  which mean the following when the attributes are of Boolean type:

  ```
  is_something == true && is_something_else == false
  ```

Pattern Matching in the Filtering Attribute Equation

By default, the =~ and !~ filter operators use simple wildcard pattern matching (glob) with the "*" and "?" wildcards. When the -regexp option is specified for any command, the complete Posix Extended Regular Expression syntax is supported.

When you use the -regexp option, you should pay special attention to how you quote filter expressions. We recommend using rigid quoting with braces around regular expressions.

Regular expressions are always anchored; that is, the expression is assumed to begin matching at the beginning of an object name and end matching at the end of an object name. You can widen the search by adding ".*" to the beginning or end of the expression as necessary.

Refer to "Glob and Regular Expression Pattern Matching Syntax" for a complete description of the Tessent Shell pattern matching mechanism and a detailed description of the regular expression syntax.

Filter Equation Examples

This example shows different filtering attribute equations used to filter instance objects:

```
register_attribute -name Att1 -value_type boolean
register_attribute -name Att2 -value_type integer -default -1
register_attribute -name Att3 -value_type string -default ""

set_attribute_value u1 -name Att1
set_attribute_value u1 -name Att2 -value 3
set_attribute_value u1 -name Att3 -value "aba"

set_attribute_value u2 -name Att1
set_attribute_value u2 -name Att3 -value "abb"

#Matches only u1 because Att2 has default value on u2 so Att2 returns
#false
get_instance -filter {Att1 && Att2}

{u1}

#Matches both because Att3 starts with a in both
get_instance -filter {Att1 && Att3 =~ "a*"}

{u1 u2}

#Matches u2 because Att2 defaults to -1.
get_instances -filter {Att2 < 2}

{u2}
```

Glob and Regular Expression Pattern Matching Syntax

The introspection commands perform pattern matching in two distinct ways with subtle usage differences. The first way is with the *name_patterns* argument and it is specified as a list of patterns to match against object names. The second one is the use of the =~ and ~! operators within a filter attribute equation.

When specifying name_patterns, the hierarchy separator (/) is used to automatically split the matching patterns so that each pattern can be matched against the leaf name of each level of hierarchy separately. For example, the command “get_instance a*/b*” means to match the leaf instance starting with b in each instance whose name starts with a. Refer to section “Matching Hierarchical Names With Escaped Identifiers” for ways to deal with that in the presence of escape identifiers, especially when the escaped identifiers contains slashes.

When specifying the filter attribute equation, the / is treated as a literal within the pattern and the entire value of the attribute is searched by the entire pattern. For example, the command “get_instance -filter {name =~ “a*/b*”}” would match instance a1/b1 as well as a1/c1/b1 because a* matches “1/c1” as the / is just another literal character to match.

In regular expressions syntax, most characters are treated as literals. That is, they match only themselves (e.g., a matches “a”). The exceptions are called meta characters and are listed in Table 4-3.

Table 4-3. Posix Extended Regular Expression Metacharacters

Meta Character	Description
.	The dot matches any single character.Within bracket expressions, the dot character matches a literal dot. For example: a.c matches “abc” and so on but [a.c] matches only “a”, “.”, or “c”.
[ ]	A bracket expression. Matches a single character that is contained within the brackets. For example, [abc] matches “a”, “b”, or “c”. [a-z] specifies a range which matches any lowercase letter from “a” to “z”. These forms can be mixed: [abcx-z] matches “a”, “b”, “c”, “x”, “y”, or “z”, as does [a-cx-z]. The - character is treated as a literal character if it is the last or the first character within the brackets: [abc-], [-abc]. Note that backslash escapes are not allowed. The] character can be included in a bracket expression if they are the first character: []abc]. The [character can appear anywhere.
[^]	Matches a single character that is not contained within the brackets. For example, [^abc] matches any character other than “a”, “b”, or “c”. [^a-z] matches any single character that is not a lowercase letter from “a” to “z”. As above, literal characters and ranges can be mixed.

Table 4-3. Posix Extended Regular Expression Metacharacters

Meta Character	Description
^	Matches the starting position of the string. In Tessent shell, all matching is rooted which means the ^ character is inferred at the beginning of every expression when it is not already specified. For example, abc.* is equivalent to ^abc.* and both mean to match any string starting with abc.
$	Matches the ending position of the string. In Tessent shell, all matching are rooted which means that the $ character is inferred at the end of every expression when it is not already specified. For example, .*abc is equivalent to .*abc$ and both mean to match any string ending with abc.
( )	Defines a marked expression. The string matched within the parentheses can be recalled later (see next entry, \n). A marked expression can also be used to specify a sequence of characters that must be matched some number of times by following it by a *, + or {m.n} sequence. For example, (ab)* matches “”, “ab”, “abab”, “ababab”, and so on.
\n	Matches what the nth marked expression matched, where n is a digit from 1 to 9. For example (ab).*\1 matches any string that starts and ends with either a or b, such as aa, aca, aba, or bab but not acb and bca. A second example (ab)(cd).*\2\1 matches acffca but not acffaa because the second marked expression matched c and the first matched a.
*	Matches the preceding element zero or more times. For example: ab*c matches “ac”, “abc”, “abbbc”, and so on. [xyz]* matches “”, “x”, “y”, “z”, “zx”, “zyx”, “xyzzy”, and so on. (ab)* matches “”, “ab”, “abab”, “ababab”, and so on.
?	Matches the preceding element zero or one time. For example: ba? matches “b” or “ba”.
+	Matches the preceding element one or more times. For example: ba+ matches “ba”, “baa”, “baaa”, and so on.
|	The choice operator matches either the expression before or the expression after the operator. For example: abc|def matches “abc” or “def”.
{m,n}	Matches the preceding element at least m and not more than n times. For example: a\{3,5\} matches only “aaa”, “aaaa”, and “aaaaa”. (ab){2,3} matches “abab” and “ababab”.

Table 4-4 the escapes character that have special meaning in Posix regular expression syntax:

Table 4-4. Special Characters in Regular Expressions

Escaped character	Description
\f	Form Feed
\n	New Line
\r	Carriage Return
\t	Tab

Table 4-5 lists the pre-defined special character sets that can be used in regular expressions:

Table 4-5. Special Characters Set in Regular Expression

Symbol	Content	Description
[:alnum:]	[A-Za-z0-9]	Alphanumeric characters
[:alpha:]	[A-Za-z]	Alphabetic characters
[:blank:]	[\t]	Space and tab
[:cntrl:]	[\x00-\x1F\x7F]	Control characters
[:digit:]	[0-9]	Digits
[:graph:]	[\x21-\x7E]	Visible characters
[:lower:]	[a-z]	Lowercase letters
[:print:]	[\x20-\x7E]	Visible characters and space character
[:punct:]	[\]\[!#$%&'()*+,./:;<=>?@\^_`{|}~-]	Punctuation characters
[:space:]	[\t\r\n\v\f]	Whitespace characters
[:upper:]	[A-Z]	Uppercase letters
[:xdigit:]	[A-Fa-f0-9]	Hexadecimal digits

Example 1

The following list of examples illustrates the use of meta characters in matching expressions:

- .at matches any three-character string ending with “at”, including “hat”, “cat”, “bat”.
- [hc]at matches “hat” and “cat”.
- [^b]at matches all strings matched by .at except “bat”.
- ^[hc]at matches “hat” and “cat”, but only at the beginning of the string or line.
- [hc]at$ matches “hat” and “cat”, but only at the end of the string or line.

- \[.\] matches any single character surrounded by “[” and “]” since the brackets are escaped. For example: “[a]” and “[b]”.
- [hc]+at matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, and so on, but not “at”.
- [hc]?at matches “hat”, “cat”, and “at”.
- [hc]*at matches “hat”, “cat”, “hhat”, “chat”, “hcat”, “ccchat”, “at”, and so on.
- cat|dog matches “cat” or “dog”.
- [[:digit:]]+ matches “1”, and “32” but not “a1”.

Example 2

The following example matches all leaf instance names between 3 and 4 characters in length.

get_instances -filter { leaf_name =~ “[[:alnum:]_]{3,4}”} -regexp

```
{u1/u2/abc u1/u3/ab_1}
```

Example 3

The following example matches all instances having a leaf instance name with a prefix matching the value within a Tcl variable followed by 1 or many digits. In the example, the attribute equation cannot be enclosed in braces because it would prevent the Tcl shell from replacing the Tcl variable. Instead, the attribute equation is enclosed in double quotes and all special characters within the expression are escaped.

set prefix “MyString_”
get_instances -filter “leaf_name =~ \”${prefix}\[\[:digit:\]]\]+\ “” -regexp

```
{u1/u3/MySting_1 u1/u2/MySting_75}
```

Example 4

The following example looks for all leaf instances with a prefix equal to abc inside all instances starting with u or U followed by one or more digits. Notice that the / is automatically inferred as a hierarchy separator causing the expression to be split in two and applied to each level of hierarchy separately.

get_instance {[Uu][[:digit:]]+/abc.*} -regexp
{u3/abcd U43/abc}

Matching Hierarchical Names With Escaped Identifiers

As described in "Glob and Regular Expression Pattern Matching Syntax" on page 107, the slashes inside a name_patterns argument have special meaning; they are interpreted as hierarchy separators unless they are within escaped identifiers. For example, consider an instance whose name has two escaped identifiers as part of its hierarchical name: "u1/\u2/u3 /\u4/u5 /insta". In Tessent Shell, all attribute, introspection, and editing commands can reference such a name with and without the escape characters and trailing spaces.

Note

Legacy commands (commands that existed prior to v9.6) only support names from which the escape characters and trailing spaces have been removed. However, the legacy commands are upgraded to accept collections of objects instead of a simple list of names so you can pass in directly the output of commands such as get_pins and get _instances.

When the escape identifier and trailing spaces are removed, the example path "u1/\u2/u3 /\u4/u5 /insta" becomes "u1/u2/u3/u4/u5/insta". As you can see, the four levels of hierarchy appear as six levels because the leaf instance names \u2/u3 and \u4/u5 look like two levels each.

Matching "u1/u2/u3/u4/u5/insta" against "u1/\u2/u3 /\u4/u5 /insta" requires searching for u1/\u2/u3 if no matches are found for u1/u2. You must apply this guessing approach recursively which can be expensive when the escaped identifiers contain many slashes. More importantly, pattern matching on these names is difficult because the / is interpreted as a hierarchy separator which incorrectly splits the name_patterns into the wrong patterns to match against the level of hierarchies. For these reasons, we recommend you keep the escape identifiers in the name_patterns argument to prevent escaped slashes from being interpreted as hierarchy separators.

The following two examples both successfully match the instance with name "u1/\u2/u3 /\u4/u5 /insta":

```
get_instances {{u1/\u2/u3   /\u4/u5 /insta}}
get_instances {{u1/\\u[0-9]+/u[0-9]+   /\\u[0-9]+/u[0-9]+   /insta}} -regexp
```

Because the get_instances command accepts a list of pattern names as input, two sets of braces are used to enclose the name_patterns argument. The Tcl shell performs a first evaluation to get the list items and consumes one set of braces; it performs a second evaluation on each item and consumes the second set of braces. Without two sets of braces, you would need to escape both bracket characters "[" and "]" to keep them as part of the regular expression and prevent them from being interpreted as a Tcl command. Notice how the \ that is part of the escaped identifier is escaped by a second \ when the -regexp option is used. This is because the \ is a special character in Posix regular expression syntax and it must be escaped to be treated as a literal.

In the second example, “get_instances {{u1/\\u[0-9]+/u[0-9]+ /\\u[0-9]+/u[0-9]+ /insta}} -regexp”, the *leaf instance* matching insta, inside leaf instances matching “\u[0-9]+/u[0-9]+”, inside leaf instances matching “\u[0-9]+/u[0-9]+”, inside leaf instance matching u1 is returned. Notice how the slashes within the escaped identifiers were not interpreted as hierarchy separators but as literals within the patterns.

The parser terminates the escape identifier string when it encounters a white space. For this reason, your patterns must not include any spaces; there is no reason to include spaces when matching instance path names.

If you need the Tcl shell to replace the Tcl variable within the regular expression, as in the example below, you must use double quotes instead of the inner set of braces and escape every special character. The \\ became \\\\ because the Tcl interpreter will replace \\\\ by \\, and the \\ will be interpreted as a \ literal by the regexp parser.

Having to escape all special characters is very error prone. We recommend that you use the double sets of braces as shown above unless your pattern contains Tcl variables that must be replaced by the shell as shown below.

```
set range “0-9”
get_instances {“u1/\\\\u\[$range\]+/u\[$range\]+ /\\\\u\[$range\]+/u\[$range\]+ /insta”} \
        -regexp
```

Chapter 5
Configuration-Based Specification

DftSpecification Configuration Syntax

This chapter describes the configuration data syntax used inside Tessent Shell to specify DFT components that are to be built and, optionally, inserted into a design. This chapter only covers the IjtagNetwork wrapper as it exists inside the DftSpecification wrapper. In subsequent releases, support for additional wrappers such as the OCC, MemoryBist, and BoundaryScan wrappers will be added and documented here. Eventually, it will be possible to specify a wide variety of DFT specifications, in a single wrapper or in multiple DftSpecification wrappers, to generate and insert any type of instrument in one or multiple insertion passes.

This interface is designed to be open and flexible. Not only will all of Mentor Graphic DFT instruments become available within Tessent Shell; any third-party provider (including end user in-house instruments) will be able to integrate their instrument types into the Dft specification process. This will enable chip designers to benefit from a unified yet flexible platform capable of defining and inserting test-related instruments from an infinite amount of sources into their designs.

This chapter uses the syntax conventions in Table 5-1 and Table 7-1 on page 341 when documenting wrappers and properties used in configuration data.

Table 5-1. Syntax Conventions for Configuration Files

Convention	Example	Usage
Italic	scan_in : *port_pin_name*;	An italic font indicates a user-supplied value.
Underline	wgl_type : <u>generic</u> | lsi**;**	An underlined item indicates the default value.
|	logic_level : <u>both</u> | high | low;	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	port_naming : *port_naming*, ...;	Ellipses indicate a repeatable value. The comment “// repeatable” also indicates a repeatable value.
//	// default: ijtag_so	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

DftSpecification

Syntax

```
DftSpecification(module_name,id) {
  rtl_extension               :  string;  // default: v
  gate_extension              :  string;  // default: vg
  reuse_modules_when_possible :  on | off |auto;

  IjtagNetwork {

  }
  //Other instrument types to be added in future release
}
```

Description

A wrapper that specifies the information used by the process_dft_specification command to build a series of instruments and, optionally, insert them into a design.

As shown in the syntax section above, this release only supports the IjtagNetwork specification. All instrument types supported by Tessent Shell will be migrated into this wrapper in subsequent releases. The ability to plug third-party instruments into the DftSpecification and have them fully supported by the process_dft_specification command will also be made available.

The DftSpecification wrapper is uniquely named using two strings. The first string is the *module_name* of the design the DFT specification is for. The second string is a general purpose *id* that differentiates multiple DFT specifications for a given design module where each DftSpecification is for a different insertion pass.

Parameters

- *module_name*

  A string that identifies the design that the DFT specification is for. When running the process_dft_specification command, the name of the current design must match this string unless the -no_insertion option is specified with this command; if the -no_insertion option is used, it can also be specified using the -design_name design_name option.

- *id*

  A string that identifies which insertion pass the DFT specification is to be used for. When executing the process_dft_specification command, the id argument of the command is used to specify this id used in the DftSpecification name. If only one DftSpecification wrapper for a given module_name is loaded in memory, the process_dft_specification command uses it even if the id is not specified as an option. Using either “rtl” or “gate” as the id string is a good strategy to identify the instruments inserted in the RTL and those inserted in the synthesized netlist.

- rtl_extension : *string*;

  A string that specifies the extension used when writing out files containing Verilog RTL descriptions of the created instruments. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of rtl_extension must be different than the value of gate_extension. The default value when unspecified is “v”.

- gate_extension : *string*;

  A string that specifies the extension used when writing out files containing gate-level Verilog modules. This extension is used in the synthesis script of the instruments and when writing out the concatenated netlist. The gate_extension is only relevant in the -no_rtl context. The string must start with a letter and may be followed by any number of letters, numbers, underscores, and periods. The value of gate_extension must be different than the value of rtl_extension. The default value when unspecified is “vg”.

- reuse_modules_when_possible : on | off | auto;

  A Boolean that specifies whether instrument modules should be reused if possible, or if different modules should be created for each instance. When the value is auto, it is considered to be On in the rtl context and Off in the no_rtl context. When inserting instruments into a netlist, it is customary to avoid repeated instances of any module as timing optimization requires them to be uniquified. The default value is auto.

IjtagNetwork

Syntax

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    DataInPorts {
    }
    DataOutPorts {
    }
    HostScanInterface (id) {            // repeatable
      Interface{ *DefSpec
      }
      DesignInstance(instance_name) {   // repeatable
      }
      ScanMux(id) {                     // repeatable
      }
      Sib(id) {                         // repeatable
      }
      Tdr(id) {                         // repeatable
      }
      Tap(id) {                         // repeatable
      }
    }
  }
}
```

Description

A wrapper that specifies the IJTAG network to be built and, optionally, inserted into the design.

This wrapper can be specified once for each DftSpecification wrapper. As illustrated in the syntax above, the IjtagNetwork specification consists of one DataInPorts wrapper, one DataOutPorts wrapper, and one or more HostScanInterface wrappers.

A HostScanInterface wrapper defines a scan interface below which a network of IJTAG nodes are serially accessible. The HostScanInterface/Interface wrapper is used to specify the end points of the HostScanInterface. By default, the wrapper defines a set of ports that may or may not already exist on the current design, but you can also use this wrapper to specify how to connect to an existing host interface present inside the design. For more details and example usage, see the description of the HostScanInterface/Interface wrapper. If the HostScanInterface/Interface wrapper only consists of ports on the current design, the add_icl_scan_interfaces and the set_icl_scan_interface_ports commands are issued during insertion such that, if you perform ICL extraction once you are done with insertion, the ScanInterface in the extracted ICL module will be named using the id of the HostScanInterface you specified in the DftSpecification.

The other five wrappers, DesignInstance, ScanMux, Sib, Tap, and Tdr define nodes serially connected below the HostScanInterface. They can be specified in any order, and they are all repeatable; however, their relative order defines the scan path connectivity. The scan-out of the first element declared in the HostScanInterface wrapper sources the scan-out port of the HostScanInterface. The scan-in port of the last element declared in the HostScanInterface wrapper is sourced by the scan-in port of the HostScanInterface.

The SIB, ScanMux, and TAP nodes become host interfaces to the client interfaces of the nodes declared inside them. See the description of the Sib, and ScanMux, Tap wrappers for more details. The example below illustrates the hierarchical nature of the IJTAG network.

Scan Testability of the IJTAG Network

Elements of the IJTAG network can be used to configure the circuit in its various scan modes. The example shown in Figure 5-1 illustrates the use of a TDR to generate the various scan mode signals. In this example, the portion of the network that gives access to this TDR must remain active during scan test and must remain unaffected by the scan test modes like any other test_setup hardware you may use. If the property keep_active_during_scan_test is set on a node, the host access path to the node is traced and every node in the access path is marked with the keep_active_during_scan_test attribute. The ICL modules are generated with this attribute and collected into the merged ICL file created by ICL extract. You can later use this information to declare those modules as non-scan during scan insertion, and force them to be kept in the graybox during graybox generation.

When the *id* of a Tdr wrapper is named “scan_modes”, the keep_active_during_scan_test property defaults to on for that TDR. This, in turn, forces all nodes needed to access it from the HostScanInterface to have keep_active_during_scan_test set to on.

Once you have extracted the final ICL using ICL extract, executing the command “get_icl_instances -filter keep_active_during_scan_test” will generate a collection of all ICL instances having keep_active_during_scan_test set to on. You can then use the tessent_design_instance attribute on the ICL instances to find the corresponding design instances and pass this list to your scan insertion and/or graybox generation step. For a complete list of attributes available on the ICL instance object type, see the icl_instance data model.

Parameters

None.

Example

The following example shows the DftSpecification of a simple IJTAG network having two SIBs connected in series; the first SIB hosts a design instance, and the second SIB hosts a TDR which in turn sources and observes the data signals of a second design instance. The example assumes the design already includes those two design instances and that an ICL file describing their ports exists for each of them.

The instance “u1/bottom_green” is an instance of module ModB and has a single ScanInterface described in its ICL file; its ScanInterface will be hooked up to the host interface of the red SIB.

The instance “u2/top_green” is an instance of module ModA and has four DataInPorts and two DataOutPorts described in its ICL file. The DataOutPorts of the pink TDR is connected to the DataInPorts of ModA and vice versa. See the description of the DesignInstance wrapper for information about how the ICL files are automatically searched and matched to the design instance during the set_current_design invocation.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
```

```
        Sib(orange) {
          Tdr(pink) {
            DataOutPorts {
              connection(0)    : u2/top_green/en;
              connection(3:1) : u2/top_green/mode[2:0];
            }
            DataInPorts {
              connection(0) : u2/top_green/go;
              connection(1) : u2/top_green/done;
            }
          }
        }
        Sib(red) {
          DesignInstance (u1/bottom_green) {}
        }
      }
    }
  }
```

Figure 5-1. IJTAG Network Example Schematic

Related Topics

DesignInstance

Sib

ScanMux

Tdr

DesignInstance

Syntax

```
IjtagNetwork
  HostScanInterface(id) {
    DesignInstance(instance_name) {
      scan_interface : scan_interface_name;
    }
  }
}
```

Description

A wrapper that references an instance in the design that has a ScanInterface to connect to.

The *instance_name* string is a design instance name relative to the current design. When the process_dft_specification command is executed without the -no_insertion option, the tool verifies that the specified *instance_name* exists within the design. The tool also verifies that the associated design module has a matched ICL module with at least one ScanInterface in it. If there is only one scan interface in the ICL module, you do not need to specify the scan_interface property, but if there is more than one ScanInterface, you must use the scan_interface property to identify which one to connect to.

ICL modules are automatically searched and matched to the design modules when the set_current_design command is executed. The search paths for the ICL files are specified using the set_design_sources -format icl command. The module name matching rules are specified using the set_module_matching_options command.

Parameters

- instance_name

  A string that refers to a design instance relative to the current design.

- scan_interface : *scan_interface_name;*

  A property that is used only when the referenced design instance is an instance of a module having more than one ScanInterface declared in its ICL module. You must repeat the DesignInstance wrapper for each *scan_interface_name* found in the ICL module, if you want to connect all of them to the network.

  If you have a design instance with multiple scan interfaces and they share a common scan-in port, you cannot connect them to a SIB because this would required independent scan-in ports. Instead, you must connect them all to inputs of a scan mux, so that you end up with one scan-in port and one scan-out port. This is illustrated in the example that follows.

Example

The following example illustrates connecting the ScanInterface of a DesignInstance to a SIB. It also shows how a design instance that has three scan interfaces sharing a common scan-in port is first connected to two scan multiplexers before being connected to a SIB.

The select of the multiplexers are sourced by the DataOutPorts of the orange TDR which automatically sets its output_timing setting to “ijtag_scan_selection”. For a description and impact of this setting, see the Tdr wrapper description and specifically the “Update Stage Timing” section.

This example also shows the use of ScanMux in the network. It provides a hint about the backward propagation of the select signal though the selected path of the multiplexer. For a complete description of the ScanMux and the backward propagation of the select signals, refer to the ScanMux wrapper description section.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataOutPorts {
            Count : 2;
          }
        }
        ScanMux(M1) {
          Select : tdr(orange)/DataOut(1);
          Input(0) {
            ScanMux(M0) {
              Select : tdr(orange)/DataOut(0);
              Input(0) {
                DesignInstance (u3/top_green) {
                  scan_interface : I1;
                }
              }
              Input(1) {
                DesignInstance (u3/top_green) {
                  scan_interface : I2;
                }
              }
            }
          }
          Input(1) {
            DesignInstance (u3/top_green) {
              scan_interface : I3;
            }
          }
        }
      }
      Sib(red) {
        DesignInstance (u1/bottom_green) {}
      }
    }
  }
}
```

Figure 5-2. DesignInstance Example Schematic

Related Topics

HostScanInterface

Sib

ScanMux

Tdr

HostScanInterface/Interface

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    Interface {
      design_instance : instance_name;  // default: null
      scan_interface  : id;             // default: null

      tck             : port_pin_name;  // default: ijtag_tck

      reset           : port_pin_name;  // default: ijtag_reset *DefSpec
      select          : port_pin_name;  // default: ijtag_sel *DefSpec
      shift_en        : port_pin_name;  // default: ijtag_se *DefSpec
      capture_en      : port_pin_name;  // default: ijtag_ce *DefSpec
      update_en       : port_pin_name;  // default: ijtag_ue *DefSpec
      scan_in         : port_pin_name;  // default: ijtag_si *DefSpec
      scan_out        : port_pin_name;  // default: ijtag_so *DefSpec
      reset_polarity  : active_high | active_low | auto;

      trst            : port_pin_name;  // default: trst *DefSpec
      tms             : port_pin_name;  // default: tms *DefSpec
      tdi             : port_pin_name;  // default: tdi *DefSpec
      tdo             : port_pin_name;  // default: tdo *DefSpec
      tdo_en          : port_name;      // default: tdo_en
      tdo_en_polarity : active_high | active_low | auto;

      daisy_chain_with_existing_client : on | off | auto;
    }
  }
}
```

Description

A wrapper used to customize the interface of the HostScanInterface.

By default, the HostScanInterface connects directly to ports on the current design and follows the default naming shown in the syntax summary above. Those ports may or may not already exist in the current design. If they are not already present, they are created during insertion if the design level is set to "physical_block" or "sub_block". If the design level is set to “chip”, ports need to already exist and be connected to pad cells. For more information, see the set_design_level and read_cell_library commands.

You can also specify internal pins to connect the HostScanInterface to an existing host interface already present in the design. Example 1 shows how the Interface wrapper is used to create top-level ports, and to hook up to an existing host port. Example 2 shows how to hook up a SIB to a TAP inserted by ETAssemble in the LV Flow.

When connecting to an internal pin, specifying the reset_polarity of the reset pin is required; otherwise, you will get an ICL extract violation later on if the assumed value is not consistent with the actual reset polarity. If specified as auto, reset_polarity is determined by counting the number of children with an active high and an active low reset polarity and taking the polarity of the larger count.

Parameters

- design_instance : *instance_name*;

  This property defines the HostScanInterface on a ScanInterface defined in an ICL module associated to an instance in the design. The specified *instance_name* must be the name of an instance that already exists in the design, and it must be the instantiation of a module having a corresponding ICL module.

  ICL modules are automatically searched and matched to the design modules when the set_current_design command is executed. The search paths for the ICL files are specified using the set_design_sources -format icl command. The module name matching rules are specified using the set_module_matching_options command.

  A design_instance cannot be specified when process_dft_specification is run using the -no_insertion option.

- scan_interface : *id*;

  This property specifies which ScanInterface to define the HostScanInterface on when the design_instance property points to an instance associated to an ICL module having more than one ScanInterface. If the associated ICL module only has one ScanInterface, the scan_interface property is not needed.

  If the referenced ScanInterface is a client ScanInterface, the host ScanInterface is defined in such a way that the inserted IJTAG nodes will end up in front of and in series with the referenced client ScanInterface. If the referenced ScanInterface is a host ScanInterface, the the inserted IJTAG nodes will connect to it. This varying behavior based on the fact that the ScanInterface is of type host or client is illustrated in Example 1.

- tck : *port_pin_name;*

  This property defines the port or the pin name to source the IJTAG TCKPort function. The default is “ijtag_tck”.

- reset : *port_pin_name;*

  This property defines the port or the pin name to source the IJTAG ResetPort function. The default is “ijtag_reset”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- select : *port_pin_name;*

  This property defines the port or the pin name that is to source the IJTAG SelectPort function. The default is “ijtag_sel”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- shift_en : *port_pin_name*;

  This property defines the port or the pin name that is to source the IJTAG ShiftEnPort function. The default is “ijtag_se”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- capture_en : *port_pin_name;*

  This property defines the port or the pin name that is to source the IJTAG CaptureEnPort function. The default is “ijtag_ce”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- update_en : *port_pin_name;*

  This property defines the port or the pin name that is to source the IJTAG UpdateEnPort function. The default is “ijtag_ue”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- scan_in : *port_pin_name*;

  This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “ijtag_si”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- scan_out : *port_pin_name;*

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “ijtag_so”. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- reset_polarity : active_high | active_low | <u>auto</u>;

  This property defines the polarity of the ResetPort. When set to auto, the reset_polarity is determine by counting the number of children with an active high or an active low reset port and taking the polarity of the larger count. When the reset property points to an existing reset pin, it is required to specify the reset polarity consistent with the source being used. Failing to specify the proper polarity results in a subsequent ICL extraction DRC violation. This property is only used when the HostScanInterface is for a non-TAP scan interface.

- trst : *port_pin_name*;

  This property defines the port or the pin name that is to source the IJTAG TRSTPort function. The default is “trst”. This property is only used when the HostScanInterface is for a TAP scan interface.

- tms: *port_pin_name;*

  This property defines the port or the pin name to receive the IJTAG TMSPort function. The default is “tms”. This property is only used when the HostScanInterface is for a TAP scan interface.

- tdi : *port_pin_name*;

  This property defines the port or the pin name that is to source the IJTAG ScanInPort function. The default is “tdi”. This property is only used when the HostScanInterface is for a TAP scan interface.

- tdo : *port_pin_name;*

  This property defines the port or the pin name to receive the IJTAG ScanOutPort function. The default is “tdo”. This property is only used when the HostScanInterface is for a TAP scan interface.

- tdo_en : *port_pin_name;*

  This property defines the port or the pin name to receive the tdo_en function. The default is "tdo_en". This property is only used when the HostScanInterface is for a TAP scan interface and the tdo port is not driven by an output pad buffer in which case no tdo_en port is created as the tdo_en signal is connected to the enable pin of the output pad buffer.

- tdo_en_polarity : active_high | active_low | auto;

  This property defines the polarity of the port or the pin name that receive the tdo_en function. This property is only used when the HostScanInterface is for a TAP scan interface. If the tdo port is driven by an output pad buffer, the tdo_en_polarity is inferred from the polarity of the enable on the TAP. An error is generated if the tdo_en_polarity property is specified to active_high and the enable pin if the output buffer is active_low and vice versa. No error is ever generated if specified to auto. If the tdo port is not driven by an output pad buffer, a tdo_en port is created with the specified active polarity. The polarity active_low is used if the property is specified or defaults to auto.

- daisy_chain_with_existing_client : on | off;

  A property of type Boolean. The default is off. When set to off, the source of the scan_out pin is first disconnected before hooking up the scan-out port of the network to it. When set to on, it is assumed a valid scan-out source is already connected to the specified scan-out port. The source is used as the scan-in of the newly inserted IJTAG network. Example 1 illustrates this.

Example 1

The following example has three HostScanInterface wrappers. The first wrapper called purple is used to insert a SIB called red in series with whatever is already connected to the purple HostScanInterface (the gray SIB in Figure 5-3). The second wrapper, called pink, is used to connect a SIB called orange to the Host ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass. The third wrapper, called gray, is used to connect a SIB called green in series with the Client ScanInterface of an existing instance in the design which, in this example, is a SIB inserted in an initial insertion pass.

```
DftSpecification(design1,pass2) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Interface {
        scan_in    : si;
        scan_out   : so;
        capture_en : ce;
        shift_en   : se;
        update_en  : ue;
        reset      : rstn;
        select     : sel;
        daisy_chain_with_existing_client : on;
      }
      Sib(red) {
      }
    }
    HostScanInterface(pink) {
      Interface {
```

```
          scan_in     : si;
          scan_out    : tessent_pass1_sib_first_inst/ijtag_from_so;
          capture_en  : ce;
          shift_en    : se;
          update_en   : ue;
          reset       : rstn;
          select      : tessent_pass1_sib_first_inst/ijtag_to_sel;
        }
        Sib(orange) {
        }
      }
      HostScanInterface (gray){
        Interface {
          design_instance   : tessent_pass1_sib_first_inst;
          scan_interface    : client;
        }
        Sib(green) {
        }
      }
    }
  }
```

Figure 5-3. HostScanInterface/Interface Example Schematic

Example 2

The next example shows how to connect a SIB to the TAP controller inserted by ETAssemble as part of one of the LV flows. The first DftSpecification wrapper uses the design_instance syntax. The second DftSpecification wrapper uses explicit function connections but results in the same implementation as the first one. Although the second method is more verbose, it works even when the connections are not all made to a single instance having an associated ICL module, which is a requirement for the first method.

```
DftSpecification(my_design,post_etassemble) {
  IJTAGNetwork {
    HostScanInterface(ijtag){
      Interface {
        design_instance : LVISION_JTAP_INST;
        scan_interface  : Bist4;
      }
      Sib(1) {
      }
    }
  }
}

DftSpecification(my_design,post_etassemble) {
  IJTAGNetwork {
    HostScanInterface(ijtag){
      Interface {
        scan_in        : LVISION_JTAP_INST/tdi;
        scan_out       : LVISION_JTAP_INST/fromBist4;
        select         : LVISION_JTAP_INST/bistEn4;
        capture_en     : LVISION_JTAP_INST/captureDR2Edge;
        shift_en       : LVISION_JTAP_INST/shiftDR2Edge;
        update_en      : LVISION_JTAP_INST/updateDREnable;
        reset          : LVISION_JTAP_INST/testLogicResetInv;
        reset_polarity : active_low;
        tck            : LVISION_JTAP_INST/tck;
        }
        Sib(1) {
        }
      }
    }
  }
```

Related Topics

ScanMux/Interface

Tdr/Interface

Sib/Interface

IjtagNetwork/DataInPorts

Syntax

```
IjtagNetwork {
  DataInPorts {
    count              : integer;           // default: auto
    port_naming        : port_naming, ...;  // default: ijtag_data_in[%d]
    multiplexing       : on | off | auto;
    connection(range)  : pin_name;          // repeatable
  }
}
```

Description

A wrapper that specifies data-in ports that may or may not exist as input ports on the current design.

These data-in ports can be used to source data-in ports on child design instances, or to source the select port of a ScanMux as shown in the example for this wrapper.

Parameters

- count : *integer*;

  A property to request a specific number of data-in ports. By default, the count is determined to be large enough to satisfy the *range* value of all connection properties.

- port_naming : *port_naming, ...;*

  A property that specifies the naming of the data-in ports. The port_naming value has two distinct formats: a single string using an index symbol “%d”, or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Din%2d will create Din00, Din01, Din02, and so on. Din%d and Din%1d mean the same thing. The default is “ijtag_data_in[%d]”.

  When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify “locked”, you will get exactly one data-in port called “locked” as illustrated in Example 2 of the Tdr/DataInPorts section.

- multiplexing : on | off | auto;

  A property that specifies if a multiplexer is to be added when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-in port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See Example 2 in the Tdr/DataOutPorts section for an illustration of this.

- connection*(range) : pin_name;*

  A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Example

The following example illustrates defining five data-in ports. The ports are named my_data_in04, my_data_in03, my_data_in02, my_data_in01, and my_data_in00. The port my_data_in00 is connected to the “en” pin of the u2/top_green instance. The ports my_data_in03, my_data_in02, and my_data_in01 are connected to the mode[2:0] pins of the same instance. If the count property was not specified, it would have defaulted to four, corresponding to the largest specified connection range index value of three. In this example, a count of five is specified to request an extra data-in port to be used by the select port of the ScanMux(red) used in the HostScanInterface(purple).

Notice that if this core was further integrated into a parent design and a Tdr was used to source the five data-in ports, the property output_timing(4) with a value of “ijtag_scan_selection” would need to be specified because bit 4 is used to reconfigure the scan network. For further explanation, see section “Update Stage Timing.”

The tool is currently unable to determine if the data-in port on a design instance is used to reconfigure the network within the specified design instance but this automation is planned for the future.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    DataInPorts {
      Count           : 5;
      port_naming     : my_data_in%2d;
      connection(0)   : u2/top_green/en;
      connection(3:1) : u2/top_green/mode[2:0];
    }
    DataOutPorts {
      port_naming : my_data_out[%d];
      connection(0) : u2/top_green/go;
      connection(1) : u2/top_green/done;
    }
    HostScanInterface(purple) {
      ScanMux(red) {
        select : DataIn(4);
        Input(1) {
          DesignInstance (u1/bottom_green) {}
        }
      }
    }
  }
```

```
}
```

Figure 5-4. DataInPorts Example Schematic

Related Topics

IjtagNetwork/DataOutPorts

Tdr/DataOutPorts

Tdr/DataInPorts

Tdr/DecodedSignal

IjtagNetwork/DataOutPorts

Syntax

```
IjtagNetwork {
  DataOutPorts {
    count              : integer;          // default: auto
    port_naming        : port_naming, ...; // default: ijtag_data_out[%d]
    Connection(range)  : pin_name;         // repeatable
  }
}
```

Description

A wrapper that specifies data-out ports that may or may not exist as output ports on the current design.

These data-out ports can be used to observe data-out ports on child design instances.

Parameters

- count : *integer*;

  A property to request a specific number of data-out ports. By default, the count is determined to be large enough to satisfy the largest range value of the connection properties.

- port_naming : *port_naming, ...;*

  A property that specifies the naming of the data-out ports. The port_naming value as two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d will create Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

  When using a list of port names, you control the exact naming and width of all data-out ports. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in Example 2 of the Tdr/DataOutPorts section.

- connection*(range)* : *pin_name;*

  A repeatable property that specifies the connections to make between the bits of the data-out port and the existing pins inside the design. Because data-out ports are sinks, the same bit cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Example

The following example illustrates the creation of two data-out ports. The count property is not specified so it defaults to two corresponding to the largest connection *range* index used. The port naming is specified to define a port named “my_data_out[1:0]”. Each bit is connected to a unique source.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    DataOutPorts {
      port_naming : my_data_out[%d];
      connection(0) : u2/top_green/go;
      connection(1) : u2/top_green/done;
    }
  }
}
```

Related Topics

IjtagNetwork/DataInPorts

Tdr/DataInPorts

Tdr/DataOutPorts

Tdr/DecodedSignal

ScanMux

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    ScanMux(id) {
      Interface {
      }
      parent_instance    : instance_name;
      leaf_instance_name : leaf_instance_name;
      select             : source;
      Attributes {
        attribute_name : attribute_value;
      }
      Input (integer) {                          // legal: 0 1
        keep_active_during_scan_test : boolean; // default: auto
        Sib(id) {
        }
        Tap(id) {
        }
        Tdr(id) {
        }
        ScanMux(id) {
        }
        DesignInstance(instance_name) {
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a scan multiplexer into the design. The Input(0) and Input(1) wrappers are used to defined the nodes that feed into that input of the multiplexer. If you omit an Input wrapper, the multiplexer simply bypasses the elements connected to the other input when the unspecified input is selected. See the example that follows for an illustration of this feature.

The actual schematic of the multiplexer is shown in Figure 5-5 and Figure 5-6. As you can see, the scan multiplexer is actually more than a simple multiplexer. When used in a DR scan path, it also has logic to supply a select line (output_en1 and output_en0) for each set of clients connected to the two scan inputs (input1 and input0). When used in a TAP scan path, it also has logic to supply a to_tms (to_tms1 and to_tms0) for each set of tap clients connected to the two scan inputs (input1 and input0).The output port is the multiplexed scan-out data of the clients to source the scan-in of the host or of the next node in series with the ScanMux.

The select input is sourced by a data signal which is either one of the DataInPorts of the IjtagNetwork, one of the DataOutPorts of a Tdr or a Tap, or an output port on an existing design instance. The select input is used to control the multiplexer and to create two gated versions of the enable_in or tms inputs. The gated versions are used as the select or tms source of the clients connected to each scan input. The enable_in port is sourced by the host of the ScanMux. If the host is a Sib, the enable_in is sourced by the to_select port. If the host is another ScanMux, the

enable_in port is sourced by one of the “enable_out#” ports. If the host is the HostScanInterface, the enable_in is sourced by the select port. The tms port is sourced by the host of the ScanMux. If the host is a ScanMux, the tms port is sourced by one of the “to_tms#” ports. If the host is the HostScanInterface, the tms is sourced by the tms port.

When the reuse_modules_when_possible property is set or inferred to be on in the DftSpecification wrapper, the ScanMux module names are hardcoded to <design_name>_<design_id>_tessent_scanmux_# where # is an integer incrementing from 1. A unique module is created for each combination of port names in the interface wrapper, attributes in the Attributes wrapper, and the value of the keep_active_during_scan_test property.

When the reuse_modules_when_possible property is set or inferred to be off in the DftSpecification wrapper, the ScanMux module names are hardcoded to <design_name>_<design_id>_tessent_scanmux_<id>.

Figure 5-5. ScanMux Internal Schematic When Used on a DR Scan Path

Figure 5-6. ScanMux Internal Schematic When Used on a TAP Scan Path

Parameters

- *id*

  A unique id that contains only letters, numbers, and underscores. Every ScanMux wrapper must have a different *id* inside a common IjtagNetwork wrapper.

- parent_instance : *instance_name;*

  A string property that specifies the parent instance in which to instantiate the scan mux module. If this property is not unspecified, the scan mux is instantiated in the root module. You can use this property to push a scan multiplexer down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, a warning is displayed during validation and a uniquify_instances command is automatically issued as part of the insertion process to uniquify the instance_name.

  To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name*;

  A string property that specifies the leaf instance name of the scan multiplexer. The property defaults to "<design_name>_<dft_spec_id>_tessent_scanmux_<scanmux_id>_inst" when unspecified. The *dft_spec_id* string is the *id* specified in the DftSpecification wrapper, and the *scanmux_id* is the *id* of the ScanMux wrapper. Tessent Shell checks for conflicts before instantiating the scan multiplexer and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- select : *source*;

  A string property that specifies the source of the select. The source must match one of the formats described in this table.

Format	**Description**
DataIn(*integer*)	Specifies that the source of the select is one of the DataInPorts of the IjtagNetwork wrapper. The *integer* value must be between *count* -1 and 0 where *count* is a property that specifies the number of DataInPorts.
Tdr(*id*)/DataOut(*integer*)	Specifies that the source of the select is one of the DataOutPorts of a Tdr. The *integer* value must be between *count* -1 and 0 where *count* is a property that specifies the number of DataOutPorts on the Tdr(*id*).
Tdr(*id*)/DecodeSignal(*name*)	Specifies that the source of the select is one of the DecodedSignal of a Tdr. The *name* value must be the name of a decoded signal on the Tdr(*id*).
Tap(*id*)/DataOut(*integer*)	Specifies that the source of the select is one of the DataOutPorts of a Tap. The *integer* value must be between *count* -1 and 0 where *count* is a property that specifies the number of DataOutPorts on the Tap(*id*).
DesignInstance(*inst_name*)/*port_name*	Specifies that the source of the select is a port on the module with the instance name *inst_name*. During ICL extraction, this pin must be or must trace to a DataOutPort or a ToSelect port of an ICL module.

- Attributes/*attribute_name* : *attribute_value*;

  A data wrapper that defines arbitrary attributes in the ScanMux ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have

registered the attribute att1 on the icl_module object type, and you have added the attribute att1 with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

- Input(0|1)/keep_active_during_scan_test : on | off | auto;

  A property that specifies that all clients connected to the ScanMux input are to be kept active during scan test modes. You should rarely need to use this property because when it is auto (default), it is automatically set to on if a Tdr with keep_active_during_scan_test set to on traces to it. See section "Scan Testability of the IJTAG Network" on page 117 for information about when to use this property.

Example

The following example illustrates the use of scan multiplexers in the DR scan paths of the IJTAG network. See Figure 5-13 for an example use in the TAP scan path. The red mux has client nodes specified in both of its Input wrappers which make it a selection multiplexer. The pink mux only has client nodes in its "1" Input wrapper making it a bypass multiplexer. The clients connected to the "1" input are bypassed when the select is low. For both multiplexers, a data-out bit on the black TDR is used as the source of the select signal.

The AND gates shown at the bottom of Figure 5-7 illustrate the distributed gating mechanism used to generate the select signals for the multiplexed TDRs. These AND gates correspond to the AND gates shown in Figure 5-5.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(black) {
        DataOutPorts {
          count : 2;
        }
      }
      ScanMux(red) {
        select : tdr(black)/DataOut(1);
        Input(1) {
          Tdr(red) {
            length : 3;
          }
        }
        Input(0) {
          ScanMux(pink) {
            select : tdr(black)/DataOut(0);
            Input(1) {
              Tdr(green) {
                length : 4;
              }
            }
          }
        }
      }
      Tdr(orange) {
      }
    }
  }
```

```
}
```

Figure 5-7. ScanMux Example Schematic

Related Topics

DesignInstance

Tdr

Sib

HostScanInterface

ScanMux/Interface

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    ScanMux(id) {
      Interface {
        select       : port_name; // default: mux_select
        input0       : port_name; // default: mux_in0
        input1       : port_name; // default: mux_in1
        output       : port_name; // default: mux_out

        enable_in    : port_name; // default: enable_in
        enable_out0  : port_name; // default: enable_out0
        enable_out1  : port_name; // default: enable_out1

        tms          : port_name; // default: tms
        to_tms0      : port_name; // default: to_tms0
        to_tms1      : port_name; // default: to_tms1
        tdo_en       : port_name; // default: tdo_en
        from_tdo_en0 : port_name; // default: from_tdo_en0
        from_tdo_en1 : port_name; // default: from_tdo_en1
      }
    }
  }
}
```

Description

A wrapper that specifies the name of the ports when creating the ScanMux module.

All port names can be specified by the user; however, they can only be simple scalar port names.

Parameters

- select : *port_name;*

  This property defines the name of the port labelled “select” in Figure 5-5 and Figure 5-6. The default is “mux_select”.

- input0 : *port_name;*

  This property defines the name of the port labelled “input0” in Figure 5-5 and Figure 5-6. The default is “mux_in0”.

- input1 : *port_name;*

  This property defines the name of the port labelled “input1” in Figure 5-5 and Figure 5-6. The default is “mux_in1”.

- output : *port_name;*

  This property defines the name of the port labelled “output” in Figure 5-5 and Figure 5-6. The default is “mux_out”.

- enable_in : *port_name;*

  This property defines the name of the port labelled “enable_in” in Figure 5-5. The default is “enable_in”.

- enable_out0 : *port_name;*

  This property defines the name of the port labelled “enable_out0” in Figure 5-5 and Figure 5-6. The default is “enable_out0”.

- enable_out1 : *port_name;*

  This property defines the name of the port labelled “enable_out1” in Figure 5-5. The default is “enable_out1”.

- tms : *port_name;*

  This property defines the name of the port labelled “tms” in Figure 5-6. The default is “tms”.

- to_tms0 : *port_name;*

  This property defines the name of the port labelled “to_tms0” in Figure 5-6. The default is “to_tms0”.

- to_tms1 : *port_name;*

  This property defines the name of the port labelled “to_tms1” in Figure 5-6. The default is “to_tms1”.

- tdo_en : *port_name;*

  This property defines the name of the port labelled “tdo_en” in Figure 5-6. The default is “tdo_en”.

- from_tdo_en0 : *port_name;*

  This property defines the name of the port labelled “from_tdo_en0” in Figure 5-6. The default is “from_tdo_en0”.

- from_tdo_en1 : *port_name;*

  This property defines the name of the port labelled “from_tdo_en1” in Figure 5-6. The default is “from_tdo_en1”.

Example

The following example illustrates the use of this wrapper to specify custom port names for the scan multiplexer used in a DR Scan Path.

```
IjtagNetwork {
   HostScanInterface(id) {
      ScanMux(id) {
         Interface {
            select      : sm_select;
            input0      : sm_si0;
            input1      : sm_si1;
            output      : sm_so;
            enable_in   : sm_en;
```

```
                enable_out0 : sm_en0;
                enable_out1 : sm_en1;
            }
        }
    }
}
```

Related Topics

HostScanInterface/Interface

Tdr/Interface

Sib/Interface

Sib

Syntax

```
IjtagNetwork
  HostScanInterface(id) {
    Sib(id) {
      Interface {
      }
      parent_instance          : instance_name;
      leaf_instance_name       : leaf_instance_name;
      keep_active_during_scan_test: boolean;           // default: auto
      capture_value            : enum;              // legal  : 1 0 self
      Attributes {
        attribute_name : attribute_value;
      }
      Sib(id) {
      }
      Tdr(id) {
      }
      ScanMux(id) {
      }
      DesignInstance(instance_name) {
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a SIB (Scan Inclusion Bit) into the IJTAG network.

A SIB is a special node in IJTAG that contains a single bit shift register. When the bit is zero, the SIB bypasses the client nodes connected to its host port. When the bit is one, the SIB includes the client nodes connected to its host port as part of the active scan path. Figure 5-8 shows the detailed schematic of its implementation.

The shift register value is latched into the update stage on the falling edge of TCK when update_en is active. A delay stage is then used to delay the transition of to_select_en by one TCK cycle so as to avoid timing races that will exist if TCK is not perfectly balanced between the SIB and its clients. Refer to section "Update Stage Timing" for more details about this feature.

When the reuse_modules_when_possible property is set or inferred to be on in the DftSpecification wrapper, the SIB module names are hardcoded to <design_name>_<design_id>_tessent_sib_# where # is an integer incrementing from 1. A unique module is created for each combination of port names in the interface wrapper, attributes in the Attributes wrapper, and the value of the capture_value and keep_active_during_scan_test properties.

When the reuse_modules_when_possible property is set or inferred to be off in the DftSpecification wrapper, the SIB module names are hardcoded to <design_name>_<design_id>_tessent_sib_<id>.

Figure 5-8. SIB Internal Schematic

As described in the create_dft_specification command description, the instruments that need to remain active during scan are inserted under one SIB, and the ones which are scan tested, such as a memory BIST controller, are inserted under another SIB, This SIB is identified with a tessent_dft_function attribute set to “scan_tested_instrument_host”. When this attribute is defined as such, the SIB is equiped with the added logic shown in Figure 5-9 such that the IJTAG network which is scan tested is fully exercised by the scan test.

Figure 5-9. Added Scan Isolation Logic for SIB(sti)

Parameters

- *id*

  A unique id that contains only letters, numbers, and underscores. Every Sib wrapper must have a different id value inside the IjtagNetwork wrapper.

- parent_instance : *instance_name*;

  A string property that specifies the parent instance in which to instantiate the SIB module. If unspecified, the SIB is instantiated in the root module. You can use this property to push a SIB down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, a warning is displayed during validation and a uniquify_instances command is automatically issued as part of the insertion process to uniquify the instance_name.

  To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a

single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name*;

  A string property that specifies the leaf instance name of the SIB. It defaults to "<design_name>_<dft_spec_id>_tessent_sib_<sib_id>_inst" when unspecified. The *dft_spec_id* string is the *id* specified in the DftSpecification wrapper, and the *sib_id* is the *id* of the SIB. Tessent Shell checks for conflicts before instantiating the SIB and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | auto;

  A property that specifies that all nodes serially connected to the SIB are to be kept active during scan test modes. For information about when to use this property, see the section "Scan Testability of the IJTAG Network" on page 117.

- capture_value : 0 |1 | self;

  The property controls what the shift register captures during the capture cycle. By default it captures a constant 0 which provides a predictable value that can always be compared when it comes out on the scan-out pin. Specifying "self" makes the SIB hold during capture. The multiplexer closest to the shift stage in Figure 5-8 is simply removed when using the "self" value. For the diagnostic hold time issue in the scan path, it is recommended to keep the default capture value of 0.

- Attributes/*attribute_name* : *attribute_value*;

  A data wrapper that defines arbitrary attributes in the SIB ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

Example

The following example illustrates the use of SIBs to provide discrete access to multiple parts of a complex network. There are three SIBs at the top of the network. The first SIB, which is closest to the scan-out and has an id value of "logictest_hw", is used to provide access to a TDR that is used to create the scan mode enable signals. Because the TDR id has a value of "scan_modes", the property keep_active_during_scan_test defaults to on, which in turns propagates to the three SIBs at the top of the network.

The second SIB, with an *id* value of "scanable_instruments", is used to access all instruments that will not be kept active during scan test but, instead, will be scan tested like the rest of the normal functional logic. This SIB can remain closed during the scan test modes such that the rest of the network remains stable and provides valid and undisturbed test_setup values.

The third SIB. with an *id* value of “child_cores”, is used to access a chain of SIBs. The chained SIBs independently open the scan path to each child core. The presence of these SIBs guarantees that a defective core or a powered down core can always be bypassed when accessing other cores.

```
DftSpecification(top,setup_hw) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(logictest_hw) {
        Tdr(scan_modes) {
          DecodedSignal(edt) {
            decode_values : 2'b01, 2'b10;
          }
          DecodedSignal(edt_config2) {
            decode_values : 2'b10;
          }
          DecodedSignal(multi_bypass) {
            decode_values : 2'b11;
          }
          DecodedSignal(scan_test_en) {
            decode_values : 2'b01, 2'b10, 2'b11;
          }
        }
      }
      // Everything below this Sib is scan tested.
      // This sibs remains off when setting up logictest_hw
      sib(scanable_instruments) {
        sib(mbist1) {
        }
        sib(mbist2) {
        }
      }
      sib(child_cores) {
        sib(c1) {
          DesignInstance(corea_i1) {
          }
        }
        sib(c2) {
          DesignInstance(corea_i2) {
          }
        }
        sib(c3) {
          DesignInstance(coreb_i1) {
          }
        }
      }
    }
  }
}
```

Figure 5-10. SIB Example Schematic

Related Topics

DesignInstance

HostScanInterface

ScanMux

Tdr

Sib/Interface

Syntax

```
IjtagNetwork {
   HostScanInterface (id) {
      Sib(id) {
         Interface {
            Client {
               select         : port_name; // default: ijtag_sel
               tck            : port_name; // default: ijtag_tck
               scan_in        : port_name; // default: ijtag_si
               scan_out       : port_name; // default: ijtag_so
               reset          : port_name; // default: ijtag_reset
               reset_polarity : active_low | active_high;
               shift_en       : port_name; // default: ijtag_se
               capture_en     : port_name; // default: ijtag_ce
               update_en      : port_name; // default: ijtag_ue
            }
            Host {
               to_select      : port_name; // default: ijtag_to_sel
               scan_in        : port_name; // default: ijtag_from_so
            }
         }
      }
   }
}
```

Description

A wrapper that contains a Client and Host wrapper that together fully specify the client and host ports.

The Client wrapper provides properties to fully specify the names of the eight client ports, and also contains a property to define the reset polarity of the reset port. The Host wrapper provides properties to fully specify the names of the two host ports.

Parameters

- Client/select : *port_name*;

  This property defines the name of the port labelled “ijtag_sel” in Figure 5-8. The default is “ijtag_sel”.

- Client/tck: *port_name*;

  This property defines the name of the port labelled “ijtag_tck” in Figure 5-8. The default is “ijtag_tck”.

- Client/scan_in : *port_name*;

  This property defines the name of the port labelled “ijtag_si” in Figure 5-8. The default is “ijtag_si”.

- Client/scan_out : *port_name*;

  This property defines the name of the port labelled “ijtag_so” in Figure 5-8. The default is “ijtag_so”.

- Client/reset : *port_name*;

  This property defines the name of the port labelled “ijtag_reset” in Figure 5-8. The default is “ijtag_reset”.

- Client/reset_polarity : active_low | active_high;

  This property defines the active polarity of the port labelled “ijtag_reset” in Figure 5-8. The default is “active_low”.

- Client/shift_en : *port_name*;

  This property defines the name of the port labelled “ijtag_se” in Figure 5-8. The default is “ijtag_se”.

- Client/capture_en : *port_name*;

  This property defines the name of the port labelled “ijtag_ce” in Figure 5-8. The default is “ijtag_ce”.

- Client/update_en : *port_name*;

  This property defines the name of the port labelled “ijtag_ue” in Figure 5-8. The default is “ijtag_ue”.

- Host/to_select : *port_name*;

  This property defines the name of the port labelled “ijtag_to_sel” in Figure 5-8. The default is “ijtag_to_sel”.

- Host/scan_in : *port_name*;

  This property defines the name of the port labelled “ijtag_from_so” in Figure 5-8. The default is “ijtag_from_so”.

Example

The following example illustrates fully specifying a unique name for each port.

```
IjtagNetwork {
   HostScanInterface (id) {
      Sib(id) {
         Interface {
            Client {
               select          : sel;
               tck             : tck;
               scan_in         : si;
               scan_out        : so;
               reset           : reset;
               shift_en        : se;
               capture_en      : ce;
               update_en       : ue;
            }
            Host {
```

```
                    to_select       : to_sel;
                    scan_in         : from_so;
                }
            }
        }
    }
}
```

Related Topics

HostScanInterface/Interface

Tdr/Interface

ScanMux/Interface

Tap

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
        }
        parent_instance              : instance_name;
        leaf_instance_name           : leaf_instance_name;
        keep_active_during_scan_test : on | off | auto;

        bypass_instruction_codes     : auto | binary , ...;

        Attributes {
          attribute_name : attribute_value;
        }

        DataOutPorts {
        }
        HostBscan {
        }
        DeviceIDRegister {
        }
        HostIjtag(<id>) {
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TAP controller into the IjtagNetwork.

A TAP is a special node in IJTAG. It contains an IEEE1149.1 state machine controlled by a TMS port. The TMS port is used to steer the state machine into various states and control when the capture, shift, and update cycles happen. Unlike a SIB and a TDR, which are only active in the DR States, the TAP has an instruction register that is activated in the IR States. The value in the instruction register determines which data register is selected when in the DR States. The TAP state machine is illustrated in Figure 5-11. A block diagram of a TAP controller is shown in Figure 5-12.

A TAP wrapper is only allowed to exist in a HostScanInterface or inside the Input wrapper of a ScanMux. A TAP is only allowed to be in series with other TAP controllers and can never be in series with Tdr and Sib nodes even through ScanMux nodes. The name of all ports on the TAP controller are user-specifiable using the Interface wrappers. BoundaryScan host ports are present when the HostBscan wrapper is specified. The device id register is present when the DeviceIDRegister wrapper is specified. Finally, the IJTAG host ports are present when one or more HostIjtag wrappers are specified. The bypass register is required and always present.

If you are familiar with the LogicVision TAP, you will remember that it had an internal DR register used to observe status bits and control user bits and decoded signals. If you need such a register, you can and should use a Tdr wrapper inside a HostIjtag wrapper of the Tap; this will give you access to the Tdr/DataInPorts, Tdr/DataOutPorts, and Tdr/DecodedSignal wrapper syntax.

All registers operate on the rising edge of TCK except for the update stage, labelled “inst_latch” in Figure 5-12 on page 154, which operates on the falling edge of TCK.

The tap module names are hardcoded to <design_name>_<design_id>_tessent_tap_<id>.

Figure 5-11. TAP FMS Diagram

Figure 5-12. TAP Block Diagram

Parameters

- *id*

  A unique id that contains only letters, numbers, and underscores. Every Tap wrapper must have a different id value inside the IjtagNetwork wrapper.

- parent_instance : *instance_name*;

  A string property that specifies the parent instance in which to instantiate the TAP module. If unspecified, the TAP is instantiated in the root module. You can use this property to push a TAP down near the client interfaces it interacts with.

  If the specified instance_name is an instance of a repeated module, the tool issues a warning during validation and automatically issues a uniquify_instances command as part of the insertion process to uniquify the instance_name.

  To perform IJTAG network insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IITAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion process can be done by going back and forth between setup and insertion mode within a

single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name*;

  A string property that specifies the leaf instance name of the TAP. It defaults to "<design_name>_<dft_spec_id>_tessent_tap_<tap_id>_inst" when unspecified. The dft_spec_id string is the id specified in the DftSpecification wrapper, and the tap_id is the id of the Tap wrapper. Tessent Shell checks for conflicts before instantiating the TAP controller and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | auto;

  A property that specifies that all nodes serially connected to the TAP are to be kept active during scan test modes. For information about when to use this property, see the section "Scan Testability of the IJTAG Network" on page 117.

- bypass_instruction_codes : auto | binary, ... ;

  A property used to define one or many opcodes for the BYPASS instruction. The all 0 code is automatically assigned to the BYPASS instruction whether specified or not. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- Attributes/*attribute_name* : *attribute_value*;

  A data wrapper that defines arbitrary attributes in the TAP ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

Example

This example defines two TAP controllers where a DataOutPort on the primary TAP is used to control if the secondary TAP is active or not. When the DataOutPort on the primary TAP is high, the secondary TAP is in series with it. When the DataOutPort on the primary TAP is low, the secondary TAP is bypassed and it is held in run_test_idle by gating its tms port low.

```
DftSpecification(module_name,id) {
  IjtagNetwork {
    HostScanInterface(purple) {
      ScanMux(green) {
        select : Tap(main)/DataOut(0);
        Input(1) {
          Tap(secondary) {
            HostIjtag(0) {
```

```
                Tdr(pink) {
                }
              }
            }
          }
        }
        Tap(main) {
          DataOutPorts {
            count : 1;
          }
        }
      }
    }
  }
```

Figure 5-13. Example Multi-TAP Configuration Schematic

Related Topics

Tap/Interface

Syntax

```
DftSpecification(module_name,id) {

IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
         fsm_state          : port_name; // def: fsm_state[%d]
         Client {
            tdi             : port_name; // default: tdi
            tdo             : port_name; // default: tdo
            tck             : port_name; // default: tck
            tms             : port_name; // default: tms
            trst            : port_name; // default: trst
            tdo_en          : port_name; // default: tdo_en
            tdo_en_polarity : active_high | active_low | auto;
          }
          Host {
            reset          : port_name; // default: test_logic_reset
            capture_en     : port_name; // default: capture_dr_en
            shift_en       : port_name; // default: shift_dr_en
            update_en      : port_name; // default: update_dr_en
            reset_polarity : active_high | active_low;
          }
        }
      }
    }
  }
}
```

Description

A wrapper that contains a Client and Host wrapper that together fully specify the names and polarity of the client and host ports.

The Client wrapper provides properties to fully specify the names of the six client ports, and also contains a property to define the polarity of the tdo_en port. The Host wrapper provides properties to fully specify the names of the four global host ports as well as a property to define the polarity of the reset port. The rest of the host ports related to the IJTAG and boundary scan ports are specified in the Tap/HostIjtag and Tap/HostBScan wrappers respectively.

Parameters

- fsm_state : port_name;

  This property defines the name of the port labelled “fsm_state” in Figure 5-12. The specified value must end with “[%d]”. The default is “fsm_state[%d]”.

- Client/tdi : port_name;

  This property defines the name of the port labelled “tdi” in Figure 5-12. The default name when unspecified is “tdi”.

- Client/tdo : port_name;

  This property defines the name of the port labelled “tdo” in Figure 5-12. The default name when unspecified is “tdo”.

- Client/tck : port_name;

  This property defines the name of the port labelled “tck” in Figure 5-12. The default name when unspecified is “tck”.

- Client/tms : port_name;

  This property defines the name of the port labelled “tms” in Figure 5-12. The default name when unspecified is “tms”.

- Client/trst : port_name;

  This property defines the name of the port labelled “trst” in Figure 5-12. The default name when unspecified is “trst”. This port is always active low.

- Client/tdo_en : port_name;

  This property defines the name of the port labelled “tdo_en” in Figure 5-12. The default name when unspecified is “tdo_en”. Its active polarity is controlled by the tdo_en _polarity property. The port is active in the Shift-DR and Shift-IR states. See Figure 5-11 for a diagram of the state machines.

- Client/tdo_en_polarity : active_high | active_low;

  This property controls the polarity of the tdo_en port. The default is active_low. When the DftSpecification is created by the create_dft_specification command, the property is automatically specified to match the polarity of the enable pin for the output pad buffer associated to the tdo port.

- Host/reset : port_name;

  This property defines the name of the port labelled “reset” in Figure 5-12. The default name when unspecified is “test_logic_reset”. Its active polarity is defined using the reset_polarity property. This port is active when the TAP fsm is in the TestLogicReset state which can be synchronously entered by keeping the tms port high for a minimum of 5 TCK cycles or asynchronously entered by setting the trst port to zero.

- Host/capture_en : port_name;

  This property defines the name of the port labelled “capture_en” in Figure 5-12. The default name when unspecified is “capture_dr_en”.

- Host/shift_en : port_name;

  This property defines the name of the port labelled “shift_en” in Figure 5-12. The default name when unspecified is “shift_dr_en”.

- Host/update_en : port_name;

  This property defines the name of the port labelled “update_en” in Figure 5-12. The default name when unspecified is “update_dr_en”.

- Host/reset_polarity : active_high | active_low;

  This property defines the active polarity of the reset port.

Example

This example defines all client ports with a “client_” prefix and all host ports with a “host_” prefix. The polarity of tdo_en and reset are both specified as active high.

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        Interface {
          Client {
            tdi             : client_si;
            tdo             : client_so;
            tck             : client_tck;
            tms             : client_tms;
            trst            : client_reset;
            tdo_en          : client_so_en;
            tdo_en_polarity : active_high;
          }
          Host {
            reset          : host_reset;
            capture_en     : host_capture_en;
            shift_en       : host_shift_en;
            update_en      : host_update_en;
            reset_polarity : active_high;
          }
        }
      }
    }
  }
}
```

Related Topics

Tap/DataOutPorts

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DataOutPorts {
          count       : int | auto;
          port_naming : port_name, ... ; // default: user_ir_bits[%d]
          multiplexing : on | off | auto;
          connection(range) : pin_name ; // repeatable
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the number (count) of data-out ports to create on the TAP module, the naming of the ports, the connections to make from them, and whether multiplexing is to be inserted when making the connections.

We do not recommend using the TAP as a source of DataOut ports. Instead, you should use the Tdr/DataOutPorts wrapper as a source. It is good practice to include a data-out port on a TAP to control the select of a ScanMux that inserts one or many secondary TAP controllers in series with the main one as illustrated in Figure 5-13. Notice that the property output_timing which exists in the Tdr/DataOutPorts wrapper is not present. There is no risk of races here because the TAP data-out ports change in the Update-IR state while all client registers are only reacting to the Update-DR state. When using the data-out port to control the insertion or exclusion of a TAP controller, the TMS signal is clocked on the rising edge of TCK, and the data-out signal changes on the falling edge of TCK thereby removing any risk of races.

Parameters

- count : *integer*;

  A property that specifies the number of data-out bits needed on the TAP. By default, the count is determined to be large enough to satisfy the range value of all connection properties.

- port_naming : *port_naming*;

  A property that specifies the naming of the data-out ports. The port_naming value has two distinct formats: a single string using an index symbol, or a list of port names with explicit sizes.

  When using the indexed format, you specify “base_name[%d]” when you want to use a bus and you specify “base_name%<integer>d” when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the

integer is padded with 0s to be at least <integer> bits wide. For example, “Dout%2d” will create Dout00, Dout01, Dout02, and so on. “Dout%d” and “Dout%1d” mean the same thing. The default is “user_ir_bits[%d]”.

When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly 7 data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in Example 2 of the Tdr/DataOutPorts.

- multiplexing : `on | off | auto;`

  A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See this illustrated in Example 2 of the Tdr/DataOutPorts wrapper.

- connection`(range)` : `pin_name`;

  A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the “process_dft_specification -no_insertion” option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Example 1

The “Example” on page 155 illustrates the creation of a data-out port with no connection property. The connection to the data-out port is done using the select property of the ScanMux as illustrated in Figure 5-13.

Example 2

The following example illustrates the specification of four data-out ports with connections to pins on a design instance.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tap(orange) {
        DataOutPorts {
          connection(0)    : u2/top_green/en;
          connection(3:1) : u2/top_green/mode[2:0];
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataOutPorts

Tap/DeviceIDRegister

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DeviceIDRegister {
          Interface {
            version_code       : port_name; //def: version_code[%d]
            part_number_code   : port_name; //def: part_number_code[%d]
          }
          capture_source       : internal | external;

          instruction_codes    : auto | binary, ... ;

          version_code         : binary; // default: 4'h0
          manufacturer_id_code : binary; // default: 11'h0
          part_number_code     : binary; // default: 16'h0
        }
      }
    }
  }
}
```

Description

A wrapper that specifies that the TAP controller is to be equipped with a device id register.

Properties are used to specify the version id, manufacturer id, and the part number codes. When the property capture_source is set to external, the version and part_number codes are captured from ports on the TAP controller. The port names are user-specifiable using the Interface wrapper. The specified version and part_number codes are used as tie values on the version and part_number code ports when instantiating the TAP controller inside the design.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Parameters

- Interface/version_code : *port_name*;

  This property defines the name of the port labelled “version_code” in Figure 5-12. The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “version_code[%d]”.

- Interface/part_number_code : port_name;

  This property defines the name of the port labelled “part_number_code” in Figure 5-12. The specified value must include the “[%d]” or “%<integer>d” symbol. The default is “part_number_code[%d]”.

- capture_source : internal | external;

  A property that defines where the capture value of the part number and version codes comes from. When the property is set to external, the version_code and part_number_code properties are captured from ports on the TAP controller. The version_code and part_number_code properties are used as tie values on the version and part_number ports when instantiating the TAP controller inside the design. When the property is set to internal, no ports are created and the capture value is describes in the RTL description of the TAP. The default value is external.

- instruction_codes : auto | binary, ... ;

  A property used to define one or many opcodes for the DEVICE_ID instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- version_code : binary;

  A property that specifies a version id code using a 4-bit binary or hexadecimal number. The default value is “4'h0”.

- manufacturer_id_code : binary;

  A property that specifies a manufacturer id code using an 11-bit binary or hexadecimal number. The default value is “11'h0”.

- part_number_code : binary;

  A property that specifies a part number code using a 16-bit binary or hexadecimal number. The default value is “16'h0”.

Example

This example specifies that the TAP controller is to have a device id register. The version and part number codes are encoded as tie values on ports of the TAP controller.

```
DftSpecification (module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        DeviceIDRegister {
          version_code      : 4'h0;
          manufacturer_id_code : 11'hA3;
          part_number_code     : 16'h0105;
        }
      }
    }
  }
}
```

Related Topics

Tap/HostBScan

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        HostBscan {
          Interface {
            select              : port_name; // def: host_bscan_to_sel
            force_disable       : port_name; // def: force_disable
            select_jtag_input   : port_name; // def: select_jtag_input
            select_jtag_output  : port_name; // def: select_jtag_output
            extest_pulse        : port_name; // def: extest_pulse
            extest_train        : port_name; // def: extest_train
            scan_in             : port_name; // def: host_bscan_from_so
          }
          InstructionCodes {
            CLAMP               : auto | unused |binary, ... ;
            EXTEST              : auto | binary, ... ;
            EXTEST_PULSE        : auto | unused | binary, ... ;
            EXTEST_TRAIN       : auto | unused | binary, ... ;
            INTEST             : auto | unused | binary, ... ;
            SAMPLE_PRELOAD     : auto | binary, ... ;
            HIGHZ              : auto | unused | binary, ... ;
          }
        }
      }
    }
  }
}
```

Description

The presence of this wrapper specifies that the TAP is to be equipped with a port capable of hosting a boundary scan chain.

The interface wrapper is used to change the default port naming of the various functions.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Parameters

- Interface/select : *port_name*;

  This property defines the name of the port labelled “bscan_select” in Figure 5-12. The default is “host_bscan_to_sel”.

- Interface/force_disable : *port_name*;

  This property defines the name of the port labelled “force_disable” in Figure 5-12. The default is “force_disable”.

- Interface/select_jtag_input : *port_name*;

  This property defines the name of the port labelled “select_jtag_input” in Figure 5-12. The default is “select_jtag_input”.

- Interface/select_jtag_output : *port_name*;

  This property defines the name of the port labelled “select_jtag_output” in Figure 5-12. The default is “select_jtag_output”.

- Interface/extest_pulse : *port_name*;

  This property defines the name of the port labelled “extest_pulse” in Figure 5-12. The default is “extest_pulse”. This port is not created when the EXTEST_PULSE property inside the Tap/HostIjtag wrapper is specified as “unused”.

- Interface/extest_train : *port_name*;

  This property defines the name of the port labelled “extest_train” in Figure 5-12. The default is “extest_train”. This port is not created when the EXTEST_TRAIN property inside the Tap/HostIjtag wrapper is specified as “unused”.

- Interface/scan_in : *port_name*;

  This property defines the name of the port labelled “from_bscan_so” in Figure 5-12. The default is “host_bscan_from_so”.

- InstructionCodes/CLAMP : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the CLAMP instruction. All codes must differ from the other codes by at least one non-X bit. When left as auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST : auto | binary, ... ;

  A property used to define one or many opcodes for the EXTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified.

- InstructionCodes/EXTEST_PULSE : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the EXTEST_PULSE instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/EXTEST_TRAIN : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the EXTEST_TRAIN instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan

wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/INTEST : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the INTEST instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

- InstructionCodes/SAMPLE_PRELOAD : auto | binary, ... ;

  A property used to define one or many opcodes for the SAMPLE_PRELOAD instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified.

- InstructionCodes/HIGHZ : auto | unused | binary, ... ;

  A property used to define one or many opcodes for the HIGHZ instruction. All codes must differ from the other codes by at least one non-X bit. When left to auto, the tool automatically selects a unique code and describes it in the ICL file when the Tap/HostBScan wrapper is specified. When specified as unused, the instruction is not included in the TAP controller.

Example

This example specifies that the TAP is to be equipped with a boundary scan host port. The port naming is left unspecified so the default naming is used. It specifies explicit opcodes for the required instructions related to boundary scan and makes a few of the optional ones unused. Notice that the all 0 opcode is not used as it is reserved for the bypass register. This example assumes no Tap/HostIjtag and no Tap/DataOutPorts wrappers are specified as there would be no room for those opcodes.

```
DftSpecification(mychip,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Tap(main) {
        bypass_instruction_codes : 2'b00;
        HostBscan {
          InstructionCodes {
            CLAMP          : unused;
            EXTEST         : 2’b01;
            EXTEST_PULSE   : unused;
            EXTEST_TRAIN   : unused;
            IDCODE         : 2’b10;
            INTEST         : unused;
            SAMPLE_PRELOAD : 2'b11;
            HIGHZ          : unused;
          }
        }
      }
    }
```

```
        }
    }
```

Related Topics

Tap/HostIjtag

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tap(id) {
        HostIjtag(id) {
          instruction_codes : auto | binary, ... ;
          Interface {
            select  : port_name; // default: host_%s_to_sel
            scan_in : port_name; // default: host_%s_from_so
          }
          Sib(id) {
          }
          Tdr(id) {
          }
          ScanMux(id) {
          }
          DesignInstance(instance_name) {
          }
        }
      }
    }
  }
}
```

Description

A repeatable wrapper that specifies the presence of one or many IJTAG host ports on the TAP controller.

The optional properties allow you to specify a specific opcode to access each IJTAG host port, and to name the ports. You can then specify any other IJTAG node types inside it.

The tool automatically assigns the instruction codes when you specify auto for all instructions. If you want to specify specific binary values, you must specify them for all instructions.

Parameters

- id

  A string that uniquely identifies the HostIjtag wrapper. The string contains only letters, numbers, and underscores.

- instruction_codes : auto | binary, ...;

  A property that defines one or many opcodes for the instruction that selects the IJTAG host port. All codes must differ from the other codes specified in the other wrappers such as inside the Tap/HostBScan wrapper by at least one non-X bit and the width must be the same. When left to auto, the tool automatically selects a unique code and describes it in the ICL file.

- Interface/select : *port_name*;

  This property defines the name of the port labelled “hostX_select” in Figure 5-12. The specified value may include the %s symbol which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_to_sel”.

- Interface/scan_in : *port_name*;

  This property defines the name of the port labelled “hostX_from_so” in Figure 5-12. The specified value may include the %s symbol which is replaced by the id of the HostIjtag wrapper. The default is “host_%s_from_so”.

Example

This example defines two IJTAG host scan interfaces. The default opcodes and port naming are used.

```
DftSpecification(my_chip,rtl) {
  IjtagNetwork {
    HostScanInterface(tap) {
      Tap(main) {
        HostIjtag(mission) {
        }
        HostIjtag(debug) {
        }
      }
    }
  }
}
```

Related Topics

Tdr

Syntax

```
DftSpecification(module_name,id) {

  IjtagNetwork {
    HostScanInterface(id) {
      Tdr(id) {
        DataInPorts {
        }
        DataOutPorts {
        }
        Interface{
        }
        parent_instance              : instance_name;
        leaf_instance_name           : leaf_instance_name;
        keep_active_during_scan_test : boolean;      // default: auto
        length                       : integer;      // default: auto
        extra_bits_capture_value     : enum;         // legal  : 1 0 self
        reset_value                  : binary;       // default: auto
        Attributes {
          attribute_name : attribute_value;
        }
        DecodedSignal(signal_name) {
        }
      }
    }
  }
}
```

Description

A wrapper that specifies the creation and, optionally, the insertion of a TDR inside the IJTAG network.

Figure 5-15 on page 176 shows the detailed schematic of a 3-bit TDR with some data-in and data-out ports as well as some decoded output signals. When requesting data-out ports or decoded signals, the default is to always use an update stage between the shift stage and the output port. In the case of decoded signals, the decoding logic is between the shift stage and the update stage. It is possible to request that the update stage be removed in situations where it does not matter that the data bits toggle during the shift cycles. See the description of the output_timing property found in the DataOutPorts and DecodedSignal sections. On the other hand, if the output data bit is used to enable or disable parts of the IJTAG network, its output timing must be further delayed by an extra cycle to avoid the race condition described in the Update Stage Timing section that follows. The TCK clock only needs to be balanced within the TDR module but can tolerate as much as 50% of a TCK period between nodes since a negative edge retiming flip-flop is used on the scan-out port of the Tdr and Sib nodes.

The TDR module names are hardcoded to <design_name>_<design_id>_tessent_tdr_<id>.

Update Stage Timing

The IJTAG network can be seen as very similar to a 1149.1 or a 1500 network. It has, however, one extra level of freedom where a TDR bit can be used to enable or disable a part of the

network while that part of the network is also active. In 1149.1, you never need to worry about races on the select lines because you are in the IR states when changing the instruction and it is only used the next time you go to the DR states. In 1500, the WIR and the selected TDR are accessed in a mutually exclusive manner as controlled by the selectWIR signal.

In IJTAG, you do not have this restriction. If you look at the schematic in Example 5-7 on page 139, you can see that while you are loading the green TDR, you may also be loading into the black TDR a value that will take out the green TDR for the next scan load and, instead, select the red TDR. As the black, green and red TDRs all receive the same update_en signal, it is important that the select signal reaching the green TDR is still active by the time the falling edge of TCK reaches the green TDR. It is also important that the select signal reaching the red TDR is still inactive by the time the falling edge of TCK reaches the red TDR.

If these conditions are not met, the green TDR may not perform its last update and the red update stage may be prematurely disturbed. This racing condition is illustrated in Figure 5-14. The source box can be seen as the black TDR while the destination box can be seen as the green TDR in Figure 5-7 on page 139. If you have skew in the TCK clock distribution where TCK2 is delayed with respect to TCK1, the update stage value labelled S1 could change too soon and block the overlap of S1 and update_en at the destination update flip-flop. Because the enabling can be going in all directions, you either have to precisely balance your TCK clock network all over the network or do as is done in Tessent Shell and add an extra delay stage for those select signals to remove the race condition even in the presence of large TCK clock skew.

Figure 5-14. Update Stage Timing Diagram

Parameters

- *id*

  A unique id that contains only letters, numbers and underscores. Every Tdr wrapper must have a different *id* inside the IjtagNetwork wrapper.

- parent_instance : *instance_name;*

  A string property that specifies the parent instance in which to instantiate the TDR module. When left unspecified, the TDR is instantiated in the common ancestor of the DataInPorts, DataOutPorts, and DecodedSignal connections. You can use this property to push a TDR down into a different location. You can specify "." as the parent_instance to force the TDR to be placed in the root module independent of the DataInPort, DataOutPort, and DecodedSignal connections. When the TDR has no DataInPorts and DataOutPorts connections, the parent_instance property defaults to the root module.

  If the specified or inferred instance_name is an instance of a repeated module, a warning is displayed during validation and a uniquify_instances command is automatically issued as part of the insertion process to uniquify the instance_name.

  To perform IjtagNetwork insertion inside a module that is used more than once without uniquifying it, you can set the current design to that module, insert the IJTAG network in it, and refer to it with a DesignInstance wrapper at the next level up. This bottom-up insertion

process can be done by going back and forth between setup and insertion mode within a single Tessent Shell invocation. You simply change the current design using the set_current_design command in setup mode.

- leaf_instance_name : *leaf_instance_name*;

  A string property that specifies the leaf instance name of the TDR. It defaults to "<design_name>_<design_spec_id>_tessent_tdr_<tdr_id>_inst" when unspecified. The *dft_spec_id* string is the *id* specified in the DftSpecification wrapper, and the *tdr_id* string is the *id* of the TDR. Tessent Shell checks for conflicts before instantiating the TDR and, if needed, applies the uniquification suffix specified by the set_insertion_options command.

- keep_active_during_scan_test : on | off | auto;

  A property that specifies that all nodes serially connected to the TDR are to be kept active during scan test modes. For information about when to use this property, see section "Scan Testability of the IJTAG Network" on page 117. This property defaults to on when the id is named "scan_modes".

- length : *integer*;

  A property to request a specific shift register length. By default, the length is determined to be large enough to satisfy the specified DataInPorts/count, DataOutPorts/count or DecodedSignal/decode_values properties.

- extra_bits_capture_value : 0 | 1 | self;

  A property that applies to the upper bits of the TDR when the length is larger than the DataInPorts/count value. The specified values control what the extra shift register bits capture during the capture cycle. The value self means that it will hold its value.

- reset_value : *binary;*

  A property that specifies a binary number as wide as the TDR to specify the reset value of the DataOutPorts and the DecodedSignals. The shift register stage is not reset. The ICL description of the TDR automatically sets the DefaultLoadValue of the ScanRegister equal to the specified reset value such that if a TDR bit resets to 1, the DefaultLoadValue for that bit will also be set to 1.

- Attributes/*attribute_name* : *attribute_value*;

  A data wrapper that defines arbitrary attributes in the TDR ICL module. This mechanism is useful for finding special attributed modules in the concatenated ICL, at a later time. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers, and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_module object type, and you have added the att1 attribute with a value of "v1" in some of your ICL modules using the Attributes wrapper, you can later quickly find those modules using the "get_icl_modules -filter att1==v1" command.

Example

The following example illustrates the syntax used to create a TDR exactly like the one shown in Figure 5-15. Notice how the output_timing property is used to make some of the outputs unlatched and some having the extra delay stage needed for ijtag_scan_selection.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(blue) {
        length                    : 3;
        extra_bits_capture_value : self;
        DataOutPorts {
          count             : 3;
          port_naming       : Dout[%d];
          output_timing(0)  : ijtag_scan_selection;
          output_timing(1)  : normal;
          output_timing(2)  : unlatched;
        }
        DataInPorts {
          count             : 1;
          port_naming       : Din%d;
        }
        DecodedSignal(sigA) {
          decode_values     : 3'bx00;
          output_timing     : ijtag_scan_selection;
        }
        DecodedSignal(sigB) {
          decode_values     : 3'bx01;
          output_timing     : normal;
        }
        DecodedSignal(sigC) {
          decode_values     : 3'bx1x;
          output_timing     : unlatched;
        }
      }
    }
  }
}
```

Figure 5-15. Detailed View of TDR Schematic

Related Topics

HostScanInterface

DesignInstance

ScanMux

Sib

Tdr/DataInPorts

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    Tdr(id) {
      DataInPorts {
        count              : integer;          // default: auto
        port_naming        : port_naming, ...;//default: ijtag_data_in[%d]
        connection(range) : pin_name;
      }
    }
  }
}
```

Description

A wrapper that specifies the number (count) of data-in ports to create on the TDR, the naming of the ports, and the connections to make to them.

Parameters

- count : *integer;*

  A property that specifies the number of data-in bits needed on the TDR. By default, the count is determined to be large enough to satisfy the *range* value of all connection properties.

- port_naming : *port_naming, ...;*

  A property that specifies the naming of the data-in ports. The port_naming value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Din%2d will create Din00, Din01, Din02, and so on. Din%d and Din%1d mean the same thing. The default is “ijtag_data_in[%d]”.

  When using a list of port names, you control the exact naming and width of each data-in port. For example, if you specify “locked”, you will get exactly one data-in port called “locked” as illustrated in Example 2.

- connection*(range) : pin_name;*

  A repeatable property that specifies the connections to make between the bits of the data-in ports and the existing pins inside the design. Because data-in ports are sinks, the same bit cannot be connected to multiple sources. It is, therefore, illegal to repeat the connection property with overlapping range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using

the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Example 1

The following example illustrates the specification of two data-in ports with connections to pins on a design instance.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataInPorts {
            connection(0) : u2/top_green/go;
            connection(1) : u2/top_green/done;
          }
        }
      }
    }
  }
}
```

Example 2

The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called “locked”. There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

```
DftSpecification(clk_gen,bottom_up) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tdr(1) {
        DataOutPorts {
          connection(5:3) : pll/mult[2:0];
          connection(2:0) : pll/div[2:0];
          port_naming     : mux_sel, mult[2:0], div[2:0];
        }
        DataInPorts {
          connection(0) : pll/locked;
          port_naming   : locked;
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataOutPorts

Tdr/DecodedSignal

Tdr/DataOutPorts

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    Tdr(id) {
      DataInPorts {
        count              : integer;
        port_naming        : port_naming, ...; //default:ijtag_data_out[%d]
        multiplexing       : on | off | auto;
        output_timing(range) : timing_spec; // repeatable
        connection(range)    : pin_name;    // repeatable
      }
    }
  }
}
```

Description

A wrapper that specifies the number (count) of data-out ports to create on the TDR, the naming of the ports, the output_timing of the ports, and the connections to make to them.

Parameters

- count : *integer*;

  A property that specifies the number of data-out bits needed on the TDR. By default, the count is determined to be large enough to satisfy the *range* value of all connection properties.

- port_naming : `port_naming, ...;`

  A property that specifies the naming of the data-out ports. The port_naming value has two distinct formats. It can be a single string using an index symbol or a list of port names with explicit sizes.

  When using the indexed format, you specify base_name[%d] when you want to use a bus and you specify base_name%<integer>d when you want to create a set of scalar ports. In both cases, the range is <count>-1 to 0. Using <integer> in the scalar naming ensures the integer is padded with 0s to be at least <integer> bits wide. For example, Dout%2d will create Dout00, Dout01, Dout02, and so on. Dout%d and Dout%1d mean the same thing. The default is “ijtag_data_out[%d]”.

  When using a list of port names, you control the exact naming and width of each data-out port. For example, if you specify “mux_sel,mult[2:0],div[2:0]”, you will get exactly seven data-out ports where the most significant bit is called mux_sel and the least significant bit is called div[0] as illustrated in Example 2.

- multiplexing : `on | off | auto;`

  A property that specifies whether to add a multiplexer when making the connection to the specified pin_name of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added to control the

select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified pin_name of the connection properties. See Example 2 for illustration.

- output_timing (*range*) : unlatched | normal | ijtag_scan_reconfiguration;

  A repeatable property that specifies the output timing of the data-out ports. As you can see in Figure 5-15, a data-out port can be directly connected to the shift register when output timing is specified as unlatched. This corresponds to Dout[2] in Figure 5-15. A data-out port can be connected to the shift register though an update stage when output timing is specified as normal. This corresponds to Dout[1] in Figure 5-15. A data-out port can be connected to the shift register though an update stage and another delay stage when output timing is specified as ijtag_scan_reconfiguration. This corresponds to Dout[0] in Figure 5-15. For a description of when the delay stage is needed, refer to the "Update Stage Timing" section.

- connection`(range)` : `pin_name`;

  A repeatable property that specifies connections to make between bits of the data-out ports and existing pins inside the design. Because data-out ports are sources, the same bit can be connected to multiple destinations by simply repeating the connection property with the same range indexes.

  If the pin name refers to a bused pin and you are connecting to a full bus, you can omit the bus range if the process_dft_specification -no_insertion option is not used. If you are using the process_dft_specification -no_insertion switch, you must specify the full range of the destination pin because the range cannot be extracted from the design.

Example 1

The following example illustrates the specification of four data-out ports with connections to pins on a design instance. The example associated with Figure 5-15 shows the use of the output_timing property.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Sib(orange) {
        Tdr(orange) {
          DataOutPorts {
            connection(0)   : u2/top_green/en;
            connection(3:1) : u2/top_green/mode[2:0];
          }
        }
      }
    }
  }
}
```

Example 2

The following example illustrates the explicit naming of the data-in and data-out ports. The data-in port is a single bit wide and called "locked". There are 7 data-out ports. The port associated with connection(0) is div[0] and the one associated with connection(5) is mult[2].

Multiplexing is left unspecified so it defaults to auto. Because the pll/mult[2:0] and pll/div[2:0] pins have a functional source, a multiplexer is inserted and a 7th data-out bit named mux_sel is used to control the multiplexers.

```
DftSpecification(clk_gen,bottom_up) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tdr(1) {
        DataOutPorts {
          connection(5:3) : pll/mult[2:0];
          connection(2:0) : pll/div[2:0];
          port_naming     : mux_sel, mult[2:0], div[2:0];
        }
        DataInPorts {
          connection(0) : pll/locked;
          port_naming   : locked;
        }
      }
    }
  }
}
```

Figure 5-16. DataOutPort Connection Multiplexing

Related Topics

Tdr/DataInPorts

Tdr/DecodedSignal

Tdr/DecodedSignal

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    Tdr(id) {
      DecodedSignal(signal_name) {
        decode_values : binary_code, ...;
        multiplexing  : on | off | auto;
        output_timing : timing_spec;
        connection    : port_pin_name;       // repeatable
        Attributes {
          attribute_name : attribute_value;

      }
    }
  }
}
```

Description

A wrapper that specifies the creation of a decoded signal.

The decode_values property is used to specify which values of the TDR make the decoded signal active. The connection property specifies connections from the decoded signals.

Parameters

- decode_values : *binary_code*, ...;

  A property that specifies a list of binary codes that define when the decoded signal is active. The binary codes must be as wide as the TDR. The binary numbers may contain Xs as long as there is at least one 0 or one 1 in each value.

- multiplexing : *on* | *off* | *auto*;

  A property that specifies whether to add a multiplexer when making the connection to the specified *pin_name* of the connection properties. When set to auto, a multiplexer is added unless the pin is unconnected, connected to a constant, or connected to a net with no fanin. As soon as one connection needs multiplexing, an extra data-out port is added on the TDR to control the select of the multiplexers. The multiplexers are inserted in the direct parent instance of the specified *pin_name* of the connection properties. See Example 2 of the Tdr/DataOutPorts section for an illustration of this.

- output_timing : *unlatched* | *normal* | *ijtag_scan_selection*;

  A property that specifies the output timing of the decode signal port. As you can see in Figure 5-15:

  - A decoded signal is directly connected to the decoding logic when output timing is specified as unlatched; this corresponds to SigC.
  - A data-out port is connected to the decoding logic though an update stage when output timing is specified as normal; this corresponds to SigB.

- A data-out port is connected to the decoding logic though an update stage and another delay stage when output timing is specified as ijtag_scan_reconfiguration; this corresponds to SigA.

Refer to the "Update Stage Timing" section for a description of when the delay stage is needed.

- connection : *port_pin_name*;

  A repeatable property that specifies connections to make between the decoded signal and existing pins inside the design. Because decoded signals are sources, they can be connected to multiple destinations by simply repeating the connection property.

- Attributes/attribute_name : *attribute_value*;

  A data wrapper that defines arbitrary attributes on the ICL port associated with the decoded signal. This mechanism is useful for finding special attributed ports in the concatenated ICL. The attribute_name can be any identifier starting with a letter and followed by any number of letters, numbers and underscores. The attribute_value can be any string or integer. If you want to define a personal attribute and later be be able to introspect its value using get_attribute_value_list, you must have registered the attribute using the "register_attribute -object_type icl_module" command. For example, assuming you have registered the att1 attribute on the icl_port object type, and you have added the att1 attribute with a value of "v1" to some DecodedSignal ports of your TDR modules using the above Attributes wrapper, you can later quickly find those ports using the "get_icl_ports -of *module_name* -filter att1==v1" command. The icl_port attributes are also inherited by the icl_pin objects so you can also find the pins using "get_icl_pins -filter att1==v1".

Example

The following example illustrates the creation of three decoded signals with each having a different output timing. The example corresponds to the schematic shown in Figure 5-15.

```
DftSpecification(MyCore,rtl) {
  IjtagNetwork {
    HostScanInterface(purple) {
      Tdr(blue) {
        DecodedSignal(sigA) {
          decode_values    : 3'bx00;
          output_timing    : ijtag_scan_selection;
        }
        DecodedSignal(sigB) {
          decode_values    : 3'bx01;
          output_timing    : normal;
        }
        DecodedSignal(sigC) {
          decode_values    : 3'bx1x;
          output_timing    : unlatched;
        }
      }
    }
  }
}
```

Related Topics

Tdr/DataInPorts

Tdr/DataOutPorts

Tdr/Interface

Syntax

```
IjtagNetwork {
  HostScanInterface(id) {
    Tdr(id) {
      Interface {
        select : port_name;  // default: ijtag_sel
        tck    : port_name;  // default: ijtag_tck
        scan_in: port_name;  // default: ijtag_si
        scan_out: port_name;  // default: ijtag_so
        reset  : port_name;  // default: ijtag_reset
        reset_polarity: active_high | active_low;
        shift_en: port_name;  // default: ijtag_se
        capture_en: port_name;  // default: ijtag_ce
        update_en: port_name;  // default: ijtag_ue
      }
    }
  }
}
```

Description

A wrapper that provides properties to fully specify the names of the eight client ports of the TDR.

The wrapper also allows defining the active polarity of the reset port.

Parameters

- select : *port_name;*

  This property defines the name of the select port. The default is “ijtag_sel”.

- tck : *port_name;*

  This property defines the name of the tck port. The default is “ijtag_tck”.

- scan_in : *port_name;*

  This property defines the name of the scan_in port. The default is “ijtag_si”.

- scan_out : *port_name;*

  This property defines the name of the scan_out port. The default is “ijtag_so”.

- reset : *port_name;*

  This property defines the name of the reset port. The default is “ijtag_reset”.

- reset_polarity : active_high | active_low;

  This property defines the active polarity of the reset port. The default is “active_low”.

- shift_en : *port_name;*

  This property defines the name of the shift_en port. The default is “ijtag_se”.

- capture_en : *port_name;*

  This property defines the name of the capture_en port. The default is “ijtag_ce”.

- update_en : *port_name;*

  This property defines the name of the update_en port. The default is “ijtag_ue”.

Example

The following example illustrates fully specifying a unique name for each port of the TDR.

```
Tdr(id) {
  Interface {
    select     : sel;
    tck        : tck;
    scan_in    : si;
    scan_out   : so;
    reset      : reset;
    shift_en   : se;
    capture_en : ce;
    update_en  : ue;
  }
}
```

Related Topics

HostScanInterface/Interface

ScanMux/Interface

Sib/Interface

DefaultsSpecifications Configuration Syntax

This section describes the configuration data syntax used inside Tessent Shell to specify default values used by the create_dft_specification command. Tessent Shell comes with built-in default values for all configuration properties. You can use the DefaultsSpecification wrapper to change many of those default values to match your preferences and requirements.

This default-setting mechanism also comes with a hierarchy of DefaultsSpecification wrappers that enable you to have company, group, and user-level defaults. In your .tessent_startup file, you can load three configuration files: one containing a DefaultsSpecification(company) wrapper, one containing a DefaultsSpecification(group) wrapper, and one containing a DefaultsSpecification(user) wrapper. The final default value of a given property is taken from the DefaultsSpecification(user) wrapper, if the property is specified in that wrapper; it is taken from the DefaultsSpecification(group) wrapper if it is defined in that wrapper but not in the DefaultsSpecification(user) wrapper; and, it is taken from the DefaultsSpecification(company) wrapper when it is defined there but not in the DefaultsSpecification(group) and DefaultsSpecification(user) wrappers. The final default value of a given property is taken from the built-in default value if it is not specified in any of the three DefaultsSpecification wrapper.

Refer to the read_config_data command to read about how to load in configuration data. See the get_defaults_value command to read about how to access the specified defaults values from your dofile. If you prefer, you can also use the “set_defaults_value -policy” command to define your different policies instead of using configuration files.

DefaultsSpecification/DftSpecification

Syntax

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {

    rtl_extension               : string;  // default: v
    gate_extension              : string;  // default: vg
    reuse_modules_when_possible : boolean; // default: auto

    IjtagNetwork {

    }
  }
}
```

Description

A wrapper that specifies options that define the default values of the properties in the DFTSpecification wrapper.

Parameters

- rtl_extension : *string*;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper.

- gate_extension : *string*;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper.

- reuse_modules_when_possible : *boolean*;

  A string that specifies the default value for the corresponding property found inside the DftSpecification wrapper.

Example

The following example shows a GlobalOptions wrapper that specifies the rtl_extension as “vb” and the gate_extension as ”v” inside the group defaults policy. Because is is not overridden by a user defaults policy wrapper, those default values are used by the create_dft_specification command.

```
> read_config_data -from_string {
  DefaultsSpecification(user) {
    DftSpecification {
      rtl_extension   : vb;
      gate_extenstion : v;
    }
  }
}
> get_defaults_value DftSpecification/rtl_extension
vb
```

IjtagNetwork

Syntax

```
DefaultsSpecification(policy) { //Legal: company | group | user
  DftSpecification {
    IjtagNetwork {
      HostScanInterface {
        Block {
          tck             : port_name;  // default: ijtag_tck
          reset           : port_name;  // default: ijtag_reset
          select          : port_name;  // default: ijtag_sel
          shift_en        : port_name;  // default: ijtag_se
          capture_en      : port_name;  // default: ijtag_ce
          update_en       : port_name;  // default: ijtag_ue
          scan_in         : port_name;  // default: ijtag_si
          scan_out        : port_name;  // default: ijtag_so
        }
        Chip {
          tck             : port_name;  // default: tck
          tdi             : port_name;  // default: tdi
          tdo             : port_name;  // default: tdo
          tms             : port_name;  // default: tms
          trst            : port_name;  // default: trst
        }
      }
    }
  }
}
```

Description

A wrapper that is used to specify defaults value of properties found in the DftSpecification/IjtagNetwork wrapper.

The properties found in the HostScanInterface/Block wrapper are used to define the Interface of the HostScanInterface when the design level is set to “physical_block” or “sub_block” using the set_design_level command. The properties found in the HostInterface/Chip wrapper are used to define the Interface of the HostScanInterface when the design level is set to “chip”.

Parameters

- ScanInterface/Block/tck : *port_name*;

  A property that sets the default value of the tck property found inside the HostScanInterface/Interfacewrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during create_dft_specification when design level is physical_block or sub_block.

- ScanInterface/Block/reset : *port_name*;

  A property that sets the default value of the reset property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during create_dft_specification when design level is physical_block or sub_block.

- ScanInterface/Block/select : *port_name*;

  A property that sets the default value of the select property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when design level is physical_block or sub_block.

- ScanInterface/Block/shift_en : *port_name*;

  A property that sets the default value of the shift_en property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.

- ScanInterface/Block/capture_en : *port_name*e;

  A property that sets the default value of the capture_en property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.

- ScanInterface/Block/update_en : *port_name*;

  A property that sets the default value of the update_en property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.

- ScanInterface/Block/scan_in : *port_name*;

  A property that sets the default value of the scan_in property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.

- ScanInterface/Block/scan_out : *port_name*;

  A property that sets the default value of the scan_out property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is physical_block or sub_block.

- ScanInterface/Chip/tck : *port_name*;

  A property that sets the default value of the tck property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is chip.

- ScanInterface/Chip/trst : *port_name*e;

  A property that sets the default value of the trst property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The

default value is used during the create_dft_specification command when the design level is chip.

- ScanInterface/Block/tms : *port_name*;

  A property that sets the default value of the tms property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is chip.

- ScanInterface/Block/tdi : *port_name*;

  A property that sets the default value of the tdi property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is chip.

- ScanInterface/Block/tdo : *port_name*;

  A property that sets the default value of the tdo property found inside the HostScanInterface/Interface wrapper of the DftSpecification/IjtagNetwork wrapper. The default value is used during the create_dft_specification command when the design level is chip.

Example

The following wrappers define custom naming convention for the IJTAG ports that serves as the default for the entire company:

```
DefaultsSpecification(company) {
  DftSpecification {
    IjtagNetwork {
      HostScanInterface {
        Block {
          tck             : my_tck;
          reset           : my_reset;
          select          : my_select;
          shift_en        : my_shift_en;
          capture_en      : my_capture_en;
          update_en       : my_update_en;
          scan_in         : my_scan_in;
          scan_out        : my_scan_out
        }
        Chip {
          tck             : my_tck;
          tdi             : my_tdi;
          tdo             : my_tdo;
          tms             : my_tms;
          trst            : my_trst;
        }
      }
    }
  }
}
```

Related Topics

DftSpecification

get_defaults_value

set_defaults_value

read_config_data

Metadata Configuration Syntax

This section describes the configuration metadata syntax used inside Tessent Shell to define all valid configuration syntax. The metadata syntax is used to define all of the available syntax documented in the DftSpecification and DefaultsSpecification/DftSpecification wrappers. The metadata syntax is also used to create the configuration tree widgets in DFTVisualizer that are used to edit and view the configuration data in the Configuration Data window. For information about how to load configuration data in the DFTVisualizer Configuration Data window, see the add_config_tab and display_specification commands. Finally, the metadata is used by the report_config_syntax command to report the available syntax in a human-readable format that is consistent with how the syntax is documented in this reference guide.

You typically do not need to define metadata because all configuration data is fully defined in the tool. You can, however, define your own metadata to define any configuration syntax needed to automate portions of your flows. Both DFTVisualizer and the report_config_syntax command, which are driven by the metadata, will completely support your syntax.

You can also augment the metadata of the DftSpecificationfor your custom needs. The added elements will be ignored by the process_dft_specification command. You can then add your own code to access the added properties and use it for your custom needs. For example, you may want to package a fully-specified DftSpecification file for an RTL core such that the user of the core can simply add the DFT hardware by running process_dft_specification with your packaged DftSpecification file. In this file, you may want to define extra information such as clock sources with their periods. After you have done this, you can use the configuration data commands such as the get_config_value command to access the information and issue the add_clocks commands in your flow. You can also add extra information that is accessed by your process_dft_specification.post_insertion Procedure and have it automatically perform custom actions based on the captured information in the DftSpecification file.

Wrapper

Syntax

```
Partitions : partition_list; //Only on first line of file
Wrapper(wrapper_name) {

  Repeatable        : On | Off | Single;
  IgnoreUnknows     : On | Off;
  Usage             : normal | hidden | Env(EnvVar);
  ReferenceableOnly : On | Off;

  InsertFirst;
  InsertLast; // default behavior
  InsertBefore      : meta_leaf_name;
  InsertAfter       : meta_leaf_name;

  Options(Visualizer) {
    cdv_tab_prefix : string;
    DisplayString  : string;
  }

  //Everything below is repeatable and optional

  Id(name_id) {
  }

  Property(property_name) {
  }
  RepeatableProperty(property_name) {
  }
  DataWrapper(wrapper_name) {
  }
  CsvWrapper(wrapper_name) {
  }

  Reference = relative_or_absolute_meta_path;
  ReferenceWithInheritedDefault = relative_meta_path;

}
```

Description

A wrapper used in metadata to define the syntax of a wrapper that is usable in configuration data.

The presence of the Partitions property is used to define the partition in which the wrapper, and all of the elements defined within in it, are to exist.

Parameters

- *Partitions : partition_list;*

  An optional property that can only be specified on the first line of a file, and that controls in which partition the metadata is to be used when defining syntax. When unspecified, the wrappers and what is defined within them, are defined in the default partition.

The partition name can be specified as “default”, “tcd”, or “default, tcd”. The “default” partition is the one in which the DftSpecification and DefaultsSpecification/DftSpecification wrappers exist. The tcd partition is the one in which the files loaded with the read_core_descriptions command are loaded into. The tcd partition is read-only and configuration data found in it cannot be modified using configuration-editing commands such as the add_config_element command.

All existing syntax defined in the “tcd” partition within the tool is also defined in the default partition. If you absolutely need to edit a tcd file, you need to load it into the default partition using the read_config_data command, perform your edits using the editing commands, save it back to the file using write_config_data, and then load it back into the tcd partition using the read_core_descriptions command. Even if it exists in the default partition, the content of this configuration data is not used by the tool because the tool only looks for tcd data in the tcd partition.

- *wrapper_name*

  A required string that defines the name of the wrapper to create. The string must start with a letter and only include letters, numbers, and underscores. The wrapper_name must be unique across all wrapper_names for root-level wrappers inside a partition, and across all wrapper_names and property_names found within a common parent wrapper.

  When you want to define extra syntax inside an already defined wrapper, you create a wrapper that contains your new elements and has a “/” as the first character. For example, if you wanted to add a new wrapper called MyCustomData inside DftSpecification/IjtagNetwork, you would do it this way:

```
Wrapper(/DftSpecification/IjtagNetwork) {
  Wrapper(MycustomData) {
    ...
  }
}
```

- Repeatable : On | Off | Single;

  A property used to define whether the wrapper is repeatable. Both the Off and Single options mean that the wrapper is not repeatable. The difference between Off and Single is that a wrapper identified as “Repeatable : Single” always exists whether or not it is specified in the configuration data. You can get the value of properties inside a Single wrapper without having to first check if the wrapper exists or not. A “Repeatable : Off” wrapper does not exist until it is specified.

- IgnoreUnknowns : On | Off;

  A property used to specify that wrappers and properties not declared in the metadata of this wrapper are to be tolerated in this wrapper. You should never set this On in wrappers that are not purely machine generated and consumed by a machine as any typos will go unnoticed and will be silently ignored. These unknown elements will not be viewable and editable in DftVisualizer and report_config_syntax will not report their syntax. It is much safer to define all allowed syntax within a wrapper even if you choose to keep some hidden using the “Usage : hidden;” property which documented next.

- Usage : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of a wrapper. When set to normal, the wrapper is reported by report_config_syntax and visible in DFTVisualizer. When set to hidden, the wrapper is not reported by report_config_syntax and is invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands such as get_config_value if the name of the hidden element is known to the user. The "Usage : Env(*EnvVar*)" is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden.

- ReferenceableOnly : On | Off;

  A property that controls the existence of the wrapper. When set to On, this wrapper definition does not exist when it is defined. It and its children only exist when referenced using the Reference property . This option is useful to create a wrapper that defines a library of elements to be referenced from other wrappers.

- InsertFirst;

  An implicit Boolean property used to specify that the wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which wrappers with "Repeatable : Single" gets displayed in the configuration tabs inside Dft Visualizer.

- InsertLast;

  An implicit Boolean property used to specify that the wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which wrappers with "Repeatable : Single" gets displayed in the configuration tabs inside DftVisualizer.

- InsertBefore : *meta_leaf_name*;

  A property used to specify that the wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a wrapper before Wrapper(IjtagNetwork), you would specify "InsertBefore : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which wrappers with "Repeatable : Single" are displayed in the configuration tabs inside DftVisualizer.

- InsertAfter : *meta_leaf_name*;

  A property used to specify that the wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a wrapper after Wrapper(IjtagNetwork), you would specify "InsertAfter : IjtagNetwork;". This is useful when you are trying to insert extra metadata into an already existing wrapper.

The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which wrappers with “Repeatable : Single” are displayed in the configuration tabs inside DftVisualizer.

- Options(Visualizer)/cdv_tab_prefix : *string*;

  A property used to control the name of the Tab that is used by DftVisualizer when the add_config_tab command is issued on this wrapper. The specified prefix is concatenated with an integer incremented from 1 for all added configuration tabs having a common prefix.

- Options(Visualizer)/DisplayString : *string*;

  A property used to control how the wrapper is labelled in the Configuration Data window. When unspecified, it default to *wrapper_name*. This option is useful when you want to create a GUI to display configuration data and you want to name the tree nodes using sentences as oppose to the strict wrapper names.

- Reference = relative_or_absolute_meta_path;

  A property used to define an element inside the wrapper using the same definition that was used inside another wrapper. This not only allows you to keep the metadata DRY, it also enables the element to be moved or copied from one location to the other using DFTVisualizer or the “add_config_element -copy_from” and the move_config_element commands. For an element to be copied or moved from one wrapper to the next, the metadata definition in both places must be created in one place by either having both of them reference a unique definition, or by having one definition reference the other. The path may be absolute or relative such as in these two examples:

  ```
  Reference = ../Interface/scan_in;
  Reference =
  /DftSpecification/IjtagNetwork/HostScanInterface/Interface/scan_in;
  ```

- ReferenceWithInheritedDefault = *relative_meta_path*;

  A property used to define a property inside the wrapper using the same definition that was used inside another wrapper. Not only is the definition inherited, the value of the referenced property becomes the default value of the new property. To use this feature, the *relative_meta_path* can only come back down in wrappers specified with “Repeatable : Single”; this is because to be able to inherit the default value from it, the referenced property must always exist whenever the referencing property exists. For example, the path *../../Interface/scan_in* is legal because Interface is defined with “Repeatable : Single” inside the IjtagNetwork/HostScanInterface wrapper. The above example could be used to create a property called “scan_in” in a wrapper found inside HostScanInterface that defaults to the value of the “../../Interface/scan_in” property when the “./scan_in” property is not specified.

Example

The following example defines the metadata used by the Sib wrapper inside the DftSpecification/IjtagNetwork wrapper. Following is the constructed syntax reported by

report_config_syntax on the same wrapper. Notice the recursive definition of the Sib node inside the Sib node achieved by referencing itself using “Reference = ../Sib;”.

```
> report_config_data -partition meta \
       /dftspecification/ijtagnetwork/hostscaninterface/sib

Wrapper(Sib) {
  Repeatable : On;
  Id(id) {
    Type : String;
    Repeatable : Off;
    IgnoreCase : On;
    Options(Visualizer) {
      Counter : global;
      DefaultValueList : #;
    }
  }
  Wrapper(Interface) {
    Repeatable : single;
    Wrapper(Client) {
      Repeatable : single;
      Property(tck) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_tck;
        }
      }
      Property(reset) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_reset;
        }
      }
      Property(select) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_sel;
        }
      }
      Property(capture_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_ce;
        }
      }
      Property(shift_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_se;
        }
      }
      Property(update_en) {
        Value(port_name) {
          Type : String;
          DefaultValue : ijtag_ue;
        }
      }
```

```
        Property(scan_in) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_si;
          }
        }
        Property(scan_out) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_so;
          }
        }
        Property(reset_polarity) {
          Options(Visualizer) {
            BlankSpaceAbove : on;
          }
          Value(active_polarity) {
            Type : String;
            LegalString : active_high, active_low;
            DefaultValue : active_low;
            IgnoreCase : On;
          }
        }
      }
      Wrapper(Host) {
        Repeatable : single;
        Property(select) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_to_sel;
          }
        }
        Property(scan_in) {
          Value(port_name) {
            Type : String;
            DefaultValue : ijtag_from_so;
          }
        }
      }
    }
    Reference = ../ScanMux/Attributes;
    Reference = ../ScanMux/parent_instance;
    Reference = ../ScanMux/leaf_instance_name;
    Reference = ../ScanMux/Input/keep_active_during_scan_test;
    Property(capture_value) {
      Value(enum) {
        type : string;
        LegalString : 1, 0, self;
        DefaultValue : 0;
      }
    }
    Reference = ../Sib;
    Reference = ../Tdr;
    Reference = ../ScanMux;
    Reference = ../DesignInstance;
  }

  > report_config_syntax \
```

```
        /dftspecification/ijtagnetwork/hostscaninterface/sib

DftSpecification(<module_name>,<id>) {
  IjtagNetwork {
    HostScanInterface(<id>) {
      Sib(<id>) {
        Interface {
          Client {
            tck            : <port_name>       ; // default: ijtag_tck
            reset          : <port_name>       ; // default: ijtag_reset
            select         : <port_name>       ; // default: ijtag_sel
            capture_en     : <port_name>       ; // default: ijtag_ce
            shift_en       : <port_name>       ; // default: ijtag_se
            update_en      : <port_name>       ; // default: ijtag_ue
            scan_in        : <port_name>       ; // default: ijtag_si
            scan_out       : <port_name>       ; // default: ijtag_so

            reset_polarity : <active_polarity> ; // legal  : active_high
(active_low)
          }
          Host {
            select  : <port_name> ; // default: ijtag_to_sel
            scan_in : <port_name> ; // default: ijtag_from_so
          }
        }
        Attributes {
          <attribute> : <string> ; // repeatable
        }
        parent_instance              : <instance_name>      ;
        leaf_instance_name           : <leaf_instance_name> ;
        keep_active_during_scan_test : <boolean> ; // default: auto
        capture_value                : <enum> ; // legal  : 1 (0) self
        Sib(<id>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/Sib
        }
        Tdr(<id>) {
          // Same as /DftSpecification/IjtagNetwork/HostScanInterface/Tdr
        }
        ScanMux(<id>) {
          // Same as
/DftSpecification/IjtagNetwork/HostScanInterface/ScanMux
        }
        DesignInstance(<instance_name>) {
          // Same as
/DftSpecification/IjtagNetwork/HostScanInterface/DesignInstance
        }
      }
    }
  }
}
```

Related Topics

Property

DataWrapper

report_config_syntax

RepeatableProperty

CsvWrapper

Property

Syntax

```
Wrapper(wrapper_name) {
  Property(property_name) {
    Usage    : normal | hidden | Env(EnvVar);

    InsertFirst;
    InsertLast; //default
    InsertBefore : meta_leaf_name;
    InsertAfter  : meta_leaf_name;

    Options(Visualizer) {
      DisplayString   : string;
      BlankSpaceAbove : on;
    }

    Value(value_id) { // repeatable
    }
  }
}
```

Description

A wrapper used in metadata to define a property that is usable in configuration data.

A property exists as soon as its parent wrapper exists regardless of whether the property is specified or not.

Parameters

- Usage : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of the property. When set to normal, the property is reported by the report_config_syntax command and is visible in DFTVisualizer. When set to hidden, the wrapper is not reported by report_config_syntax and is invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands, such as the get_config_value command, if the name of the hidden property is known by the user. The “Usage : Env(*EnvVar*)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise, it is treated as hidden.

- InsertFirst;

  An implicit Boolean property used to specify that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which properties are displayed in the Configuration Option panel inside DftVisualizer.

- InsertLast;

  An implicit Boolean property used to specify that the property is to be inserted after the already existing elements in the wrapper. This is useful if you are trying to insert extra

metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which properties are displayed in the Configuration Option panel in DftVisualizer.

- InsertBefore : *meta_leaf_name*;

  A property used to specify that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which properties get displayed in the Configuration Options panel in DftVisualizer.

- InsertAfter : *meta_leaf_name*;

  A property used to specify that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which properties get displayed in the Configuration Options panel in DftVisualizer.

- Options(Visualizer)/DisplayString : *string*;

  A property used to control how the property is identified in the GUI. When unspecified, it defaults to *property_name*. This option is useful when you want to create a GUI to display configuration data and you want to name the properties using sentences as opposed to the strict property name.

- Options(Visualizer)/BlankSpaceAbove : On | Off;

  A property used to control the insertion of a blank space above the property when creating the Configuration Options panel. You can see its effect in the panel shown in Figure 5-17 on page 203. The option was specified inside the tck, reset, trst and daisy_chain_with_existing_client property wrappers.

Figure 5-17. Example Option panel where blank spaces were added

Example

The following example defines properties of all types. It is used in the example of the get_config_value command to illustrate how to introspect them. You can use report_config_data DftSpecification -partition meta to see many more examples.

```
Wrapper(SingularWrapper) {
  Repeatable : single;
  Wrapper(WrapperWithRepeatableId) {
    id(name*) {
      type : string;
    }
    Reference = /SingularWrapper/WrapperWithRepeatableId;
    Property(time_property) {
      Value(time) {
        type : time;
        DefaultValue : 3.5ns;
      }
    }
    Property(integer_property) {
      Value(integer) {
        type : int;
        DefaultValue : 3;
      }
    }
    Property(binary_property) {
      Value(binary) {
        type : binary;
        DefaultValue : 3'b0;
```

```
          }
        }
        Property(string_property) {
          Value(string*) {
            type : string;
            DefaultValue : "";
          }
        }
        CSVWrapper(CsvWrapper) {
          Repeatable : on;
          id(module_name) {
            Type : String;
          }
          id(design_id) {
            Type : String;
            IgnoreCase : On;
          }
        }
        DataWrapper(DataWrapper_with_no_value) {
          RowIdElement(port) {
            Type : String;
            repeatable : on;
          }
        }
        DataWrapper(DataWrapper_value) {
          RowIdElement(port) {
            Type : String;
            repeatable : on;
          }
          Value(period) {
            Type : time;
          }
        }
      }
    }
```

Related Topics

DftSpecification

DataWrapper

report_config_syntax

RepeatableProperty

CsvWrapper

RepeatableProperty

Syntax

```
Wrapper(wrapper_name) {
  RepeatableProperty(property_name) {
    Usage    : normal | hidden | Env(envVar);

    InsertFirst;
    InsertLast; //default
    InsertBefore : meta_leaf_name;
    InsertAfter  : meta_leaf_name;

    Options(Visualizer) {
      DisplayString   : string;
    }
    Id(id) { //repeatable and optional
    }
    Value(value_id) { //repeatable, minimum one
    }
  }
}
```

Description

A wrapper used in metadata to define a repeatable property that is usable in configuration data.

As opposed to normal properties, a repeatable property has no default values and does not exist until it is specified.

Parameters

- Usage : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of a property. When set to normal, the property is reported by report_config_syntax and visible in DFTVisualizer. When set to hidden, the wrapper is not reported by report_config_syntax and invisible in DFTVisualizer. Its value can still be fetched and edited using the configuration introspection and editing commands such as the get_config_value command if the name of the hidden property is known to the user. The “Usage : Env(*EnvVar*)” is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden.

- InsertFirst;

  An implicit Boolean property used to specify that the property is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the properties gets displayed in the Configuration Options panel in DftVisualizer.

- InsertLast;

  An implicit Boolean property used to specify that the property is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra

metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the properties gets displayed in the Configuration Options panel in DftVisualizer.

- InsertBefore : *meta_leaf_name*;

  A property used to specify that the property is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the properties get displayed in the Configuration Options panel in DftVisualizer.

- InsertAfter : *meta_leaf_name*;

  A property used to specify that the property is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a property after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the properties get displayed in the Configuration Options panel in DftVisualizer.

- Options(Visualizer)/DisplayString : *string*;

  A property used to control how the folding region associated to a repeatable property is identified in the Configuration Data window. When unspecified, it defaults to *property_name*. A convention of using lower case name with underscore is employed when naming normal and repeatable properties in the tool. The convention for the folding region name in DftVisualizer is to use Camel naming. This is achieved by using this option.

Example

The following example shows how the connection property is defined inside the Tdr wrapper.

```
> report_config_data [get_config_value Tdr/DataOutPorts/Connection \
        -in_wrapper /DftSpecification/IjtagNetwork/HostScanInterface \
        -partition meta -object]

RepeatableProperty(connection) {
  Options(Visualizer) {
    DisplayString : Connections;
  }
  Id(range) {
    Type : string;
    Repeatable : On;
  }
  Value(pin_name) {
    Type : string;
    Repeatable : On;
  }
}
```

Related Topics

DftSpecification

DataWrapper

report_config_syntax

Property

CsvWrapper

DataWrapper

Syntax

```
Wrapper(wrapper_name) {
  DataWrapper(wrapper_name) {
    Usage              : normal | hidden | Env(envVar);

    InsertFirst;
    InsertLast; //default
    InsertBefore       : meta_leaf_name;
    InsertAfter        : meta_leaf_name;

    Options(Visualizer) {
      DisplayString    : string;
    }
    RowIdElement(element_id) { // not_repeatable, required
    }
    Value(value_id) { // repeatable, optional
    }
  }
}
```

Description

A wrapper used in metadata to define a data wrapper that is usable in configuration data.

As opposed to a normal wrapper, a data wrapper contains only data rows composed of arbitrary row id element with optional values. This type of wrapper is useful to associate values to arbitrary elements such as ports:

```
ClockPeriods {
  clka : 4ns;
  clkb : 3ns;
}
```

A data wrapper is like a wrapper with “Repeatable : Single”. It is not repeatable and starts to exist as soon as its parent wrapper exists.

Parameters

- Usage : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of the data wrapper. When set to normal, the data wrapper is reported by report_config_syntax and visible in DFTVisualizer. When set to hidden, the data wrapper is not reported by report_config_syntax and invisible in DFTVisualizer. Its content can still be fetched and edited using the configuration instrospection and editing commands such as the get_config_value command if the name of the hidden data wrapper is known to the user. The “Usage : Env(*EnvVar*)” is treated as normal if the EnvVar environment variable is defined and set to 1 otherwise it is treated as hidden.

- InsertFirst;

  An implicit Boolean property used to specify that the data wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert

extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the data wrappers get displayed in the Configuration Options panel inside DftVisualizer.

- InsertLast;

  An implicit Boolean property used to specify that the data wrapper is to be inserted after all already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command; it also controls the order in which folding regions associated to the data wrappers get displayed in the Configuration Options panel inside DftVisualizer.

- InsertBefore : *meta_leaf_name*;

  A property used to specify that the data wrapper is to be inserted before an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a data wrapper before Property(scan_in), you would specify “InsertBefore : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which folding regions associated to the data wrappers get displayed in the Configuration Options panel inside DftVisualizer.

- InsertAfter : *meta_leaf_name*;

  A property used to specify that the data wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a data wrapper after Property(scan_in), you would specify “InsertAfter : scan_in;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command. Also, it controls the order in which folding regions associated to the data wrappers get displayed in the Configuration Options panel inside DftVisualizer.

- Options(Visualizer)/DisplayString : *string*;

  A property used to control how the folding region associated to a data wrapper is identified in the Configuration Data window. When unspecified, it defaults to *wrapper_name*. This option is useful when you want to create a GUI to display configuration data and you want to name the folding regions as opposed to the strict data wrapper name.

Example

The following example shows how the ClockPeriods data wrapper shown in the description section above is defined in metadata. Notice the value of type time with the extra allowed symbols sync and sync2x.

```
DataWrapper(ClockPeriods) {
  RowIdElement(clock_port) {
    Repeatable : Off;
    Type       : String;
    IgnoreCase : Off;
  }
  Value(period) {
    Type           : Time;
    AllowedSymbols : sync, sync2x;
  }
}
```

Related Topics

DftSpecification

Property

RepeatableProperty

CsvWrapper

report_config_syntax

CsvWrapper

Syntax

```
Wrapper(wrapper_name) {
  CsvWrapper(wrapper_name) {
    Usage        : normal | hidden | Env(envVar);

    InsertFirst;
    InsertLast; //default
    InsertBefore : meta_leaf_name;
    InsertAfter  : meta_leaf_name;

    Id(id) { //repeatable and optional
    }
  }
}
```

Description

A wrapper used in metadata to define a CSV wrapper that is usable in configuration data.

As opposed to a normal wrapper, a CSV wrapper contains only rows composed of arbitrary comma separated values.

```
csv {
  v1,v2,"v3 with spaces or comma";
  v4,v5;
}
```

See the command description of report_config_data and write_config_data to see how the wrapper content can be imported from or exported to Excel. CSV wrappers are not visible nor editable inside DftVisualizer. They are meant to be manipulated graphically inside Excel.

Parameters

- Usage : normal | hidden | Env(*EnvVar*);

  A property that controls the visibility of the CSV wrapper. When set to normal, the wrapper is reported by the report_config_syntax command. When set to hidden, the wrapper is not reported by report_config_syntax. Its value can still be fetched and edited using the configuration instrospection and editing commands such as get_config_value if the name of the hidden CSV wrapper is known to the user. The "Usage : Env(*EnvVar*)" is treated as normal if the EnvVar environment variable is defined and set to 1; otherwise it is treated as hidden.

- InsertFirst;

  An implicit Boolean property used to specify that the CSV wrapper is to be inserted before the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertLast;

  An implicit Boolean property used to specify that the CSV wrapper is to be inserted after the already existing elements in the wrapper. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertBefore : *meta_leaf_name*;

  A property used to specify that the CSV wrapper is to be inserted before the already existing elements in the wrapper. You refer to the element using its leaf_name. For example, to insert a CSV wrapper before Wrapper(IjtagNetwork), you would specify “InsertBefore : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

- InsertAfter : *meta_leaf_name*;

  A property used to specify that the CSV wrapper is to be inserted after an already existing element in the wrapper. You refer to the element using its leaf_name. For example, to insert a CSV wrapper after Wrapper(IjtagNetwork), you would specify “InsertAfter : IjtagNetwork;”. This is useful when you are trying to insert extra metadata into an already existing wrapper. The relative position of the elements in the metadata controls the order in which they are reported by the report_config_syntax command.

Example

The following example shows how a generic scratch CSV wrapper is defined in the tool. You can use scratch CSV wrappers to import and introspect CSV data generated by Excel as shown in Example 2 of the report_config_data command.

```
CsvWrapper(csv) {
  Repeatable : On;
  Usage : hidden;
  Id(name) {
    Type : string;
  }
}
```

Related Topics

DftSpecification

Property

RepeatableProperty

DataWrapper

report_config_syntax

Id

Syntax

```
Wrapper(wrapper_name) {
  Id (name_id) {
    Type           : String | Boolean | Real | Int | Time | Binary;
    LegalString    : string_list;        // For String
    LegalInt       : integer_range_list; // For Int
    LegalReal      : real_range_list;    // For Real
    LegalSize      : int | Unsized;      // For Binary
    AllowedSymbols : string_list;        // For all but String
    IgnoreCase     : On | Off;
    Repeatable     : On | Off;
    Mandatory      : On | Off;

    Option(Visualiser) {
      Counter          : local | global |
                         ParentWrapper(wrapper_name).var_name;
      DefaultValueList : string_list;
    }
  }
  RepeatableProperty(property_name) {
    Id (name_id) {
    }
  }
  CsvWrapper(wrapper_name) {
    Id (name_id) {
    }
  }
}
```

Description

A wrapper used in metadata to define an identifier for named wrappers, repeatable properties, and named CSV wrappers.

Parameters

- *name_id*

  A string used to name the identifier. This name can later be referenced using the "get_config_value -id *name_id*" command.

  The *name_id* must start with a letter, include only letters, numbers and underscores, and may be terminated by an asterisk "*". If the *name_id* ends with an *, the Id is repeatable. If there is more than one Id wrapper, only the last one is allowed to have an * at the end. For example:

  ```
  Wrapper(Step) {
    Id(instance*) {
    }
  }
  ```

  allows Step(u1,u2), where u1 and u2 are repeated instance Ids.

- Type : *value*;

  A property that defines the type of the Id. The allowed types are String, Boolean, Real, Int, Time, and Binary. Ids of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>.

- LegalString : *string_list*;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list will consider casing or not based on the Id’s IgnoreCase property.

- LegalInt : *integer_range_list*;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

- LegalReal : *real_range_list*;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- LegalSize : *int* | Unsized;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- AllowedSymbols : *string_list*;

  A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols will consider casing or not based on the Id’s IgnoreCase property.

- IgnoreCase : On | Off;

  A property that can be used to make the LegalString and the AllowedSymbols case-insensitive. This option also affects if casing is ignored or not when performing the repeatability check described in the Repeatable property below.

- Repeatable : On | Off;

  A property that controls if a given *name_id* can be repeated or not. The validation of the repeatability will consider casing or not based on the IgnoreCase property documented above.

- Mandatory : On | Off;

  A property used to specify if the Id is required or not. This option can only be specified Off for the second or below Id wrapper, and if the Id wrappers below it are also Off.

- Option(Visualiser)/Counter : local | global | ParentWrapper(*wrapper_name*).*var_name*;

  A property used to define how the Id will be populated automatically when adding the wrapper graphically inside DftVisualizer. When the DefaultValueList reaches the element with a # character, the # character is replaced by an integer. The integer is a counter that is either local to the wrapper, global to the entire root wrapper, or local to a named variable attached to a parent wrapper. For example, Ids defined with a counter as ParentWrapper(*controller*).*memory_interface* will share a counter when found anywhere below a wrapper called "controller". The specified parent wrapper leaf name must exist in the ancestry of the element defining the Id.

- Option(Visualiser)/DefaultValueList : string_list

  A property used to define a list of strings that will be used for the first, second, and so on instances of the wrapper. The string elements may contain any number of letters, numbers and underscores. The last element of the list must include a # character and may be a just a # character. For example, the IjtagNetwork/HostScanInterface/Id(id) is defined with a DefaultValueList of "ijtag, ijtag#". If you add a first HostScanInterface, you will see its Id will be "ijtag". Any other subsequent instances will be labelled ijtag0, ijtag1, etc.

Example

The following example defines a wrapper with two Ids, the second one is repeatable and optional.

```
Wrapper(EX1) {
  Id(name) {
    type : string;
  }
  Id(period*) {
    type : time;
    Mandatory : off;
  }
}
```

Related Topics

DftSpecification

CsvWrapper

report_config_data

RepeatableProperty

Value

Value

Syntax

```
Wrapper(wrapper_name) {

  Property(property_name) {
    Value (value_id) {
      Type           : String | Boolean | Real | Int | Time | Binary |
                       ImplicitBoolean;
      LegalString    : string_list;        // For String
      LegalInt       : integer_range_list; // For Int
      LegalReal      : real_range_list;    // For Real
      LegalSize      : int | Unsized;      // For Binary
      AllowedSymbols : string_list;        // For all but String and
                                           //   ImplicitBoolean
      IgnoreCase     : On | Off;
      Mandatory      : On | Off;
    }
  }
  RepeatableProperty(property_name) {
    Value (value_id) {
    }
  }
  DataWrapper(wrapper_name) {
    Value (value_id) {
    }
  }
}
```

Description

A wrapper used in metadata to define an identifier for values of properties, repeatable properties, and data wrappers.

Parameters

- *value_id*

  A string used to name the value. You can reference this name later by using the "get_config_value -value_id *value_id*" command.

  The *value_id* must start with a letter, include only letters, numbers and underscores and may be terminated by an asterisk "*". If the *value_id* ends with an *, the value is repeatable. If there is more than one Value wrapper, only the last one is allowed to have an * at the end. For example:

  ```
  Property(memory_instances) {
    Value(instance*) {
    }
  }
  ```

  allows "memory_instances : u1, u2;", where u1 and u2 are repeated instance values.

- Type : *value*;

  A property that defines the type of the Value. The allowed types are String, Boolean, Real, Int, Time, Binary, and ImplicitBoolean. Values of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the Verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>. ImplicitBoolean values are implied true as soon as the property is specified with no value. The property InsertFist in the Property wrapper is an example of a property with Type ImplicitBoolean.

- LegalString : *string_list*;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list will consider casing or not based on the Value’s IgnoreCase property.

- LegalInt : *integer_range_list*;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

- LegalReal : *real_range_list*;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- LegalSize : *int* | Unsized;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- AllowedSymbols : *string_list*;

  A property that can be used when Type is set to anything but String and ImplicitBoolean to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property for Value is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols will consider casing or not based on the Value’s IgnoreCase property.

- IgnoreCase : On | Off;

  A property that can be used to make the LegalString and the AllowedSymbols case-insensitive.

- Mandatory : On | Off;

  A property used to specify if the Value is required or not. This option can only be specified Off for the second or below Value wrapper, and if the Value wrappers below it are also Off.

Example

The following example defines a property with two values; the second one is repeatable and optional.

```
Property(EX1) {
  Value(name) {
    type : string;
  }
  Value(period*) {
    type : time;
    Mandatory : off;
  }
}
```

Related Topics

DftSpecification

CsvWrapper

report_config_syntax

RepeatableProperty

Value

RowIdElement

Syntax

```
DataWrapper(wrapper_name) {
  RowIdElement (element_id) {
    Type           : String | Boolean | Real | Int | Time | Binary;
    LegalString    : string_list;        // For String
    LegalInt       : integer_range_list; // For Int
    LegalReal      : real_range_list;    // For Real
    LegalSize      : int | Unsized;      // For Binary
    AllowedSymbols : string_list;        // For all but String
    IgnoreCase     : On | Off;
    Repeatable     : On | Off;
  }
}
```

Description

A wrapper used in metadata to define the syntax of the RowIdElements of a data wrapper.

Parameters

- *element_id*

  A string used to name the row id element.

  The *element_id* must start with a letter, include only letters, numbers and underscores.

- Type : String | Boolean | Real | Int | Time | Binary;

  A property that defines the type of the RowIdElement. The allowed types are String, Boolean, Real, Int, Time, and Binary. RowIdElements of type time are allowed to have a unit string: s, ms, us, ns, ps, or fs. Binary numbers can be expressed in binary or hexadecimal format using the verilog syntax: <size>‘b<binary_digits>, ‘b<binary_digits>, <size>‘h<hexadecimal_digits>, and ‘h<hexadecimal_digits>.

- LegalString : *string_list*;

  A property that can be used when Type is set to String to limit the string to an allowed list. The validation of the allowed list will consider casing or not based on the IgnoreCase property documented below.

- LegalInt : *integer_range_list*;

  A property that can be used when Type is set to Int to limit the integer to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1..9, 20, 21”.

- LegalReal : *real_range_list*;

  A property that can be used when Type is set to Real to limit the real to some allowed ranges. Use two periods in a row to denote a range. Use the comma to separate multiple ranges such as in “1.0..9.5, 20.0, 21.0”.

- LegalSize : *int* | Unsized;

  A property that can be used when Type is set to Binary to limit the binary number to a specific size.

- AllowedSymbols : *string_list*;

  A property that can be used when Type is set to anything but String to allow extra symbols along with the allowed Boolean, Real, Int, Time, and Binary values. The LegalSize property above is an example of a property of type Int with an extra allowed symbol of “Unsized”. The validation of the allowed symbols will consider casing or not based on the IgnoreCase property documented below.

- IgnoreCase : On | Off;

  A property that can be used to make the LegalString and the AllowedSymbols case-insensitive. This option also affects whether casing is ignored when performing the repeatability check described in the following Repeatable property.

- Repeatable : On | Off;

  A property that controls whether a given element_id can be repeated or not within the Data Wrapper. The validation of the repeatability will consider casing or not based on the preceding IgnoreCase property.

Example

The following example defines a data wrapper with RowIdElement which is a arbitrary string meant to represent a clock port name. Each clock port name are only allowed once as specified by the fact that Repeatable is set to Off. Because IgnoreCase is also Off, port names Clk and clk will not be considered as repeated port names.

```
DataWrapper(ClockPeriods) {
  RowIdElement(clock_port) {
    Repeatable : Off;
    Type       : String;
    IgnoreCase : Off;
  }
  Value(period) {
    Type           : Time;
    AllowedSymbols : sync. sync2x;
  }
}
```

Related Topics

DftSpecification

CsvWrapper

report_config_syntax

RepeatableProperty

Value

Chapter 6
Instrument Connectivity Language (ICL)

Instrument Connectivity Language Syntax

This chapter specifies the syntax of the Instrument Connectivity Language (ICL) of the IEEE P1687 standard draft. The syntax described here is based on the draft version 1.67. Most of the language elements of the IEEE P1687 draft are supported. Currently unsupported ICL language elements or language elements with limited support are clearly marked as follows:

Note

The DefaultLoadValue parameter is currently not supported within the ScanInterface.

The language is introduced not through the formal grammar used in the IEEE P1687 draft, but through a hierarchical description of an ICL Module and its elements. Examples and suggestions for usage are provided for each language element. These elements are divided into language statements; that is, all the elements in the direct body of the ICL Module definition (plus a few more), and parameters. Parameters to an ICL language statement further define the properties of the statement. For example, the ICL language statement ScanRegister has the parameter ResetValue, which defines which value the scan register will assume if the reset signal turns active.

The following sections introduce common features of the language.

Numbers

ICL recognizes unsigned integer numbers in any of the following three number base formats: decimal, hexadecimal, and binary.

Unsized decimal numbers may be composed of a sequence of the digits 0 to 9. If needed, their size will be determined based on context.

Sized numbers are composed of a size constant, followed by an optional white space and a base token (d or D for decimal, h or H for hexadecimal, and b or B for binary), and the digits of the number, expressed in the respective number base. The size constant always expresses the number of binary digits of the sized number, independent of the used base format. For example, the sized numbers 4’b1011, 4’hb and 4’d11 are identical and exactly 4 binary digits wide.

Binary, decimal, and hexadecimal numbers may use their respective common digit symbols (0 and 1 for binary, 0 to 9 for decimal, 0 to 9 and a to f for hexadecimal). These digit symbols are

case insensitive. In addition, binary and hexadecimal numbers may use x or X to denote the unknown digit value.

Padding is applied automatically as necessary if the number of digits of the sized number is less than the size constant requires. For example, assume this comparison:

```
LogicSignal test { dataIn[7:0] == 8'b10 ; }
```

The "==" operator compares two 8-bit (that is, 8 binary digit wide sized numbers). However, only 2 digits "10" are given. Padding is executed based on the following rules: If the left most digit is non-x, then the padding adds as many 0 to the left as necessary. If however the leftmost digit is x, then padding adds as many x to the left as necessary. The above example will therefore be padded to the following:

```
LogicSignal test { dataIn[7:0] == 8'b00000010 ; }
```

The opposite of padding is truncation. Truncation happens if the number of digits of the sized number is larger than the size constant requires. Consequently, excessive digits are removed from the left until the size requirement is matched. However, only occurrences of 0 or x may be removed. Removing any other digit symbol results in an error.

Identifiers

IEEE P1687 differentiates between four identifiers: The scalar identifier, the parameter identifier, the vector identifier, and the pin identifier.

The scalar identifier is used to denote the elements of the ICL netlist such as the ports of a module, a module name or an instance name. The scalar identifier is case sensitive and must start with a letter. Thereafter, the scalar identifier may use letters, numbers, and the underscore symbol. It may not contain any white spaces nor the "[" or "]" symbols (these are used for vector identifiers).

The parameter identifier is used to reference a Parameter or LocalParameter definition. The parameter reference is composed of the "$" symbol and the scalar identifier name of the referenced (local) parameter. For example, if the parameter scalar identifier is userValue, the the parameter reference will be $userValue.

There are two types of vector identifiers: The single index and the range vector identifier. There is no length limit for these identifiers.

A single index vector identifier is composed of a scalar identifier, followed by the "["symbol, an unsigned decimal number, followed by the "]" symbol. Its width is considered to be 1. An example is bus[4], denoting the 4th bit of the scalar identifier bus.

A ranged vector identifier is composed of a scalar identifier, followed by the "[" symbol, a first unsigned decimal number, the ":" symbol, a second unsigned decimal number, and followed by the "]" symbol. An example is:

```
bus[0:4]
```

denoting 5 bits of the scalar identifier bus. Note that the range can either be increasing, for example [0:4], or decreasing, for example [4:0].

Pin identifiers refer to a port name, optionally including an instance path name. The port name is either a scalar, a single index or range index vector identifier. The instance names are all scalar identifiers, and the instance path is composed of these scalar identifiers, separated by the "." symbol. An example is:

```
top.myInstA.myInstB.dataIn[7:0]
```

In this example the identifiers "top", "myInstA" and "myInstB" are all denoting names of instances, whereas dataIn[7:0] is a ranged vector identifier of a port scalar named dataIn.

Limited Identifier Mathematics

Parameter identifiers, single indexes and ranged indexes may use limited mathematical operations: + (addition), - (subtraction), * (multiplication), / (division), and % (modulo). For parameter identifiers, a parameter value can be computed using these operations; for vector identifiers any used unsigned decimal number can be expressed using these operations. In addition, parameter reference may be used. Examples are:

```
Parameter ten = 10 ;
Parameter twenty = 2*$ten ;
bus[$twenty:$ten]
bus[$twenty%2]
bus[$ten-1:0]
```

Concatenations

Under certain conditions, identifiers might be concatenated with other identifiers and/or sized and unsized numbers. The concatenation operator is the "," symbol. A first example is the following:

```
Parameter topBit 1'b1 ;
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, 0, 4'b1111 ; }
```

Observe that the left hand side of the comparator (==) is 8 bits wide. The right hand side has 2 sized numbers of a total width of 5 bits. Therefore the unsized number 0 in the middle will be auto-expanded into 3'b000. It is obvious that such a comparison construct may contain at most one unsigned number. This example shows also the usage of parameter references in lieu of a number.

A concatenation may also include the "~" symbol, denoting a sized, bit-wise, binary inversion. For example: ~4' ha is equivalent to 4'h5.

Interestingly, the "~"symbol can also be used within the concatenation:

```
LogicSignal test { dataInA[3:0], dataInB[3:0] == $topBit, ~0, 4'b1111 ; }
```

Here, the unsized, but inverted 0 is auto-expanded into 4'b1111.

Syntax Conventions

This chapter uses the syntax conventions in Table 6-1 when documenting ICL syntax properties.

Table 6-1. Syntax Conventions for ICL Files

Convention	Example	Usage
Upright	ScanInPort *port_name* ;	An upright font indicates an ICL language element, either a statement or a parameter.
Italic	scan_in : *port_pin_name*;	An italic font indicates a user-supplied value.
Underline	wgl_type : <u>generic</u> | lsi**;**	An underlined item indicates the default value.
|	logic_level : <u>both</u> | high | low;	The vertical bar separates a list of values from which you must choose one. Do not include the bar in the configuration file.
...	port_naming : *port_naming*, ...;	Ellipses indicate a repeatable value. The comment "// repeatable" also indicates a repeatable value.
//	// default: ijtag_so	The double slash indicates the text immediately following is a comment and tells the tool to ignore the text.

Module

Syntax

```
Module module_name {
  Attribute att_name = att_value;
  Parameter param_name = param_value;
  LocalParameter param_name = param_value;
  ScanInterface interface_name { … }
//---- Port Functions
  ScanInPort port_name { … }
  ShiftEnPort port_name { … }
  CaptureEnPort port_name { … }
  UpdateEnPort port_name { … }
  DataInPort port_name { … }
  DataOutPort port_name { … }
  SelectPort port_name { … }
  ResetPort port_name { … }
  TMSPort port_name { … }
  TCKPort port_name { … }
  ClockPort port_name { … }
  TRSTPort port_name { … }
  AddressPort port_name { … }
  WriteEnPort port_name { … }
  ReadEnPort port_name { … }
  ScanOutPort port_name { … }
  ToShiftEnPort port_name { … }
  ToUpdateEnPort port_name { … }
  ToCaptureEnPort port_name { … }
  ToResetPort port_name { … }
  ToSelectPort port_name { … }
  ToClockPort port_name { … }
  ToTCKPort port_name { … }
  ToTMSPort port_name { … }
  ToTRSTPort port_name { … }
  ToIRSelectPort port_name { … }

  Instance instance_name Of another_module_name { … }

  LogicSignal signal_name { … }
  ScanRegister register_name { … }
  DataRegister register_name { … }
  ScanMux mux_name SelectedBy selector { … }
  DataMux mux_name SelectedBy selector { … }
  ClockMux mux_name SelectedBy selector { … }
  OneHotDataGroup name { … }
  OneHotScanGroup name { … }

  Enum enum_name { … }
  Alias alias_name = element_list { … }

  AccessLink instance_name Of link_type { … }
}
```

Description

The syntax above lists the complete contents (statements) of an ICL module. Not all statements may be used at the same time.

Each ICL statement is explained in detail in the following pages. All statements are hyperlinked from here to the respective statement description, as well as back to this module description.

Parameters

- *module_name*

  The Module statement defines the name of an ICL module. The name of the module must be unique among all loaded modules. To build an ICL design, you use the Instance statement.

Related Topics

NameSpace
UseNameSpace
Instance

Attribute

Example Syntax

```
Module module_name {
  Attribute att_name = att_value ;
}
```

Description

The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool.

Tessent understands several port attributes and pin attributes as defined in this manual. However, Tessent does not have built-in Attributes for all ICL objects. For example, the ScanInterface has no Tessent attribute.

Note that most Tessent attributes are read-only and are automatically generated by the Tessent tool, which also assigns a value to the attribute.

The Attribute statement is not only allowed in Module, but in most ICL objects documented in this Chapter.

Note

In its usage within ICL objects, like modules, ports, scan or data mux, etc., the Attribute statement is always optional.

Parameters

- *att_name* = *att_value*

  An required string and optional “=”symbol and value combination, where *att_name* defines the name of the Attribute, and *att_value*, if present, defines the value of the attribute.

  Within an ICL object where multiple Attribute statements are used, each *att_name* must be unique.

Example 1

The first example shows a very simple usage of assigning a string “true” to an attribute name “is_top”:

```
Module top {
  Attribute is_top = “true” ;
  [...]
}
```

Example 2

The second example shows a Tessent attribute `connection_rule_option` that influences ICL Network Extraction. Please see Influencing ICL Extraction through ICL Module Attributes in the *Tessent IJTAG User’s Manual* for the details of the attribute.

```
DataInPort din {Attribute connection_rule_option = "allowed_tied_high";}
```

ScanInterface

Syntax

```
Module module_name {
  ScanInterface interface_name {
    Attribute att_name = att_value;
    Port port_name;
    DefaultLoadValue load_value;
    Chain chain_name { ... }
  }
}
```

Description

The definition of a set of ports of the module that together perform a scan operation.

The ScanInterface defines a set of ports of the module that together perform a scan operation. A typical example of ports include the shift enable, capture, and update enable ports, together with at least a pair of scan-in and scan-out ports. However, the ScanInterface is applied also to the TAP, since a shift-based interface to or from a module is also defined; that is, a scan interface. See the discussion on *port_names* below, which defines the sets of ports of the four recognized use models of the ScanInterface — the client interface, the host interface, the client TAP interface, and the host TAP interface. A module may have more than one scan interface, in particular several client interfaces or interfaces for different use models. However, only one scan interface may be active at the same time. This means, for example, that a set of scan chains that are supposed to be operated together must all be in the same ScanInterface definition.

For a handoff module, with more than one portfunction of type ScanInPort or ScanOutPort, there must be as many ScanInterface statements as there are scan interfaces in the module. This does not mean, for example, that the number of scan-in and scan-out port pairs defines the number of ScanInterface statements. Since several pairs of scan-in and scan-out ports may belong to the same scan interface, which means they are supposed to be operated simultaneously, the number of ScanInterface statements may be smaller.

The examples below are for simple ScanInterface definitions. Additional examples, especially those for multiple parallel scan chains, are provided in the ScanInterface/Chain section.

Parameters

- *interface_name*

  A required string that defines the name of the scan interface of the module.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent does not have any attributes for the ICL ScanInterface statement.

- Port *port_name*

  A repeatable, optional keyword-string pair that references the name of an existing port of the module. Four use models of a ScanInterface can be distinguished. The port_names for a given ScanInterface must all belong to the same use model.

  - The client interface — Through a client scan interface a module receives all signals necessary to operate a defined set of scan chains. Ports of the following port functions form a client scan interface: at least one pair of ScanInPort and ScanOutPort, one ShiftEnPort and/or at least one SelectPort, and at most one CaptureEnPort, UpdateEnPort, ResetPort, or TCKPort. If the shift enable, capture enable, or update enable ports are not stated and the module has exactly one of each, these ports are implied for the scan interface using the modules set of port functions, or for non-handoff modules, the set or implied set of port functions. If more than one pair of scan-in and scan-out ports belong to the same scan interface, the Chain object must be used. In this case, no port name should reference a scan-in and scan-out port, nor should the DefaultLoadValue be used.

    If a non-handoff module has exactly one client interface, than the ScanInterface statement is not required. The tool assumes a “default” client interface.

  - The host interface — Through a host scan interface a module defines shift operating control signals for another module. Ports of the following port functions form a host scan interface: one ScanInPort and/or one ScanOutPort, one ToShiftEnPort and/or at least one ToSelectPort, and at most one ToCaptureEnPort, ToUpdateEnPort, ToResetPort, or ToTCKPort. Similar to the client interface, if the module has exactly one ToShiftEnPort, ToCaptureEnPort, or ToUpdateEnPort, these ports are assumed for each host scan interface, for which they have not been stated already.

  - The client TAP interface — Through a client TAP interface, the module receives all signals necessary to operate a TAP controller: One of each ScanInPort, ScanOutPort, and TMSPort, and at most one of TRSTPort and TCKPort.

  - The host TAP interface — Through a host TAP interface, the module defines signals for another module’s client TAP interface: One ScanInPort and/or one ScanOutPort, one ToTMSPort, and at least one of ToTRSTPort and ToTCKPort.

- DefaultLoadValue *load_value*

  An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. This keyword is currently not supported within the ScanInterface.

- Chain *chain_name*

  A repeatable, optional keyword-string pair that declares the name of a scan chain of the module. All chain names must be unique within a ScanInterface. If the ScanInterface has only one ScanInPort/ScanOutPort pair, than Chain object is not required.

Example 1

The following example shows three equivalent description of an ICL module with a single scan-in / scan-out client scan interface. This is not a complete list, other variations are possible. Note that for the module tdr4_1, no ScanInterface statement is given. It is not necessary, since the module has exactly one client scan interface. However, you may provide a ScanInterface as shown for modules tdr4_2, and tdr4_3.

```
Module tdr4_1 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}

Module tdr4_2 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanInterface interface1 {
     Port en ;
     Port si; Port so;
   }

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}

Module tdr4_3 {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanInterface interface2 {
     Port se ;
     Chain chain1 {
        Port si; Port so;
     }
```

```
      }

      ScanRegister R[7:0] {
         ScanInSource     si;
         ResetValue       8'b0 ;
      }
   }
```

Example 2

The next example shows the port functions and both the client and host TAP interfaces of a typical TAP controller module.

```
   Module tap {
      TCKPort         tck;
      ScanInPort      tdi;
      ScanOutPort     tdo       {  Source IRMux; }
      TMSPort         tms;
      TRSTPort        trst;
      ToSelectPort    tdrEn     {  Source sel;  }
      ScanInPort      fromTdr;
      ToSelectPort    tapEn     {  Source IR[2]; }
      ToCaptureEnPort tce;
      ToShiftEnPort   tse;
      ToUpdateEnPort  tue;

      ScanInterface clientTAP {
        Port tdi; Port tdo;
        Port tms; Port trst;
      }
      ScanInterface hostTAP   {
        Port fromTdr1; Port tdrEn1;
        Port tce; Port tse; Port tue;
      }
      [...]
   }
```

Example 3

The next example shows an example implementation of a SIB (Segment Insertion Bit) module. This module implements a scan chain switch by selecting either of two scan input ports. The module also generates an outgoing enable signal that is correlated to the second scan-in port.

```
   Module sib {
      ScanInPort       si;
      ScanOutPort      so   { Source SIB; }
      ShiftEnPort      se;
      CaptureEnPort    ce;
      UpdateEnPort     ue;
      SelectPort       en;
      TCKPort          tck;
      ScanInPort       fso;
      ToSelectPort     ten  { Source And; }

      ScanInterface client {
        Port si; Port so;
        Port se; Port ce; Port ue; Port en;
      }
```

```
      ScanInterface host   {
        Port fso; Port ten;
      }

      ScanRegister SIB {
         ResetValue     1'b0;
         ScanInSource   Mux;
      }

      ScanMux Mux SelectedBy SIB {
         1'b0 : si;
         1'b1 : fso;
      }

      LogicSignal And { en,SIB == 2'b11;  }
   }
```

Related Topics

Chain
ScanInPort
SelectPort
ShiftEnPort
CaptureEnPort
UpdateEnPort
ResetPort
TCKPort
TMSPort
TRSTPort
ScanOutPort
ToShiftEnPort
ToCaptureEnPort
ToUpdateEnPort
ToResetPort
ToTCKPort
ToTMSPort
ToTRSTPort
ScanRegister
ScanMux
LogicSignal

Chain

Syntax

```
ScanInterface interface_name {
  Chain chain_name {
    Attribute att_name = att_value;
    Port port_name;
    DefaultLoadValue load_value;
  }
}
```

Description

The Chain statement defines the name of a scan chain within a ScanInterface.

This statement does not create a scan register in the sense of the ScanRegister statement. Here it is only a declaration which ScanInPort and ScanOutPort of the module will be operated together with the control signals of the ScanInterface. There can be more than one Chain statement in a ScanInterface. In this case, each scan chain name must be unique.

Parameters

- Chain *chain_name*

  A repeatable, required keyword-string pair that defines the name of a scan chain within a given ScanInterface. There can be more than one scan chain in any given ScanInterface. However, the names of the scan chains within a ScanInterface must all be unique.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent does not have any built-in attributes for the ScanInterface/Chain statement.

- Port *port_name*

  A required keyword-string pair, with the port_name referencing a ScanInPort or ScanOutPort port of the module. Within each Chain statement, there are exactly two Port statements, one referencing the scan-in port, and the other referencing the scan-out port.

- DefaultLoadValue *load_value*

  An optional keyword-string pair that defines the value to be loaded into the single scan chain defined by a pair port names of portfunction type ScanInPort and ScanOutPort of a black-box module. This keyword is currently not supported within the Chain object.

Example 1

The following example shows a module with two client ScanInterfaces, each operating two scan chains.

```
Module chip1 {
   ScanInPort  siA1 ;
```

```
      ScanInPort  siA2 ;
      ScanOutPort soA1 { Source blockA.so1 ; }
      ScanOutPort soA2 { Source blockA.so2 ; }

      ScanInPort  siB1 ;
      ScanInPort  siB2 ;
      ScanOutPort soB1 { Source blockB.so1 ; }
      ScanOutPort soB2 { Source blockB.so2 ; }

      ShiftEnPort   se1 ;
      CaptureEnPort ce1 ;
      UpdateEnPort  ue1 ;

      ShiftEnPort   se2 ;
      CaptureEnPort ce2 ;
      UpdateEnPort  ue2 ;

      TCKPort tck ;

      ScanInterface cA12 {
         Port se1; Port ce1; Port ue1;
         Chain c1 { Port siA1; Port soA1 ; }
         Chain c2 { Port siA2; Port soA2 ; }
      }

      ScanInterface cB12 {
         Port se2; Port ce2; Port ue2;
         Chain c1 { Port siB1; Port soB1 ; }
         Chain c2 { Port siB2; Port soB2 ; }
      }
      [...]
   }
```

Example 2

In this variation of Example 1 above, the ShiftEnPort, CaptureEnPort, and UpdateEnPort ports are shared among both ScanInterface. The selection between the two interfaces is accomplished through the two SelectPort sel1 and sel2, respectively.

```
   Module chip2 {
      ScanInPort  siA1 ;
      ScanInPort  siA2 ;
      ScanOutPort soA1 { Source blockA.so1 ; }
      ScanOutPort soA2 { Source blockA.so2 ; }

      ScanInPort  siB1 ;
      ScanInPort  siB2 ;
      ScanOutPort soB1 { Source blockB.so1 ; }
      ScanOutPort soB2 { Source blockB.so2 ; }

      SelectPort   selA ;
      SelectPort   selB ;

      ShiftEnPort   se ;
      CaptureEnPort ce ;
      UpdateEnPort  ue ;

      TCKPort tck ;
```

```
      ScanInterface cA12 {
         Port selA;
         Chain c1 { Port siA1; Port soA1 ; }
         Chain c2 { Port siA2; Port soA2 ; }
      }

      ScanInterface cB12 {
         Port selB;
         Chain c1 { Port siB1; Port soB1 ; }
         Chain c2 { Port siB2; Port soB2 ; }
      }
      [...]
   }
```

Example 3

This example is another variation of Example 1. The difference is that all ports are busses and not individual ports as in Example 1. The operation of the modules is identical.

```
   Module chip3 {
      ScanInPort  si[3:0] ;
      ScanOutPort so[3:0] { Source blockA.so1, blockA.so2,
                                   blockB.so1, blockB.so2 ;
                          }

      ShiftEnPort   se[1:0] ;
      CaptureEnPort ce[1:0] ;
      UpdateEnPort  ue[1:0] ;

      TCKPort tck ;

      ScanInterface cA12 {
         Port se[0]; Port ce[0]; Port ue[0];
         Chain c1 { Port si[0]; Port so[0] ; }
         Chain c2 { Port si[1]; Port so[1] ; }
      }

      ScanInterface cB12 {
         Port se[1] ; Port ce[1]; Port ue[1];
         Chain c1 { Port si[2]; Port so[2] ; }
         Chain c2 { Port si[3]; Port so[3] ; }
      }
      [...]
   }
```

Related Topics

ScanInterface

ScanRegister

ScanInPort

Syntax

```
Module module_name {
  ScanInPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a scan input port of a Module.

A module may have zero or more scan-in ports. However, if a module has at least one ScanOutPort defined, then there must be at least one ScanInPort.

If a module has exactly one ScanInPort, than the module may also have a ScanInterface defined. However, if a module has more than one ScanInPort, than there must be at least one ScanInterface definition.

The scan-in port is used to execute a serial write (that is, not a parallel write operation). If a parallel write is to be done, use the DataInPort port function instead.

Parameters

- *port_name*

  A required string that identifies the name of a scan-in port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no ScanInterface is required. The scan-in port is named “si”. Observe that the port name is reused in the body of the ScanRegister R[7:0] declaration. There, it declares that the scan register will receive its scan data from the object named “si”, which happens to be the ScanInPort of the Module.

```
Module tdr2 {
```

```
      ScanInPort       si;
      ScanOutPort      so  { Source R[0]; }
      ShiftEnPort      se;
      SelectPort       en;
      CaptureEnPort    ce;
      UpdateEnPort     ue;
      TCKPort          tck;

      ScanRegister R[7:0] {
         ScanInSource     si;
         ResetValue       8'b0 ;
         DefaultLoadValue 8'b0 ;
      }
   }
```

Related Topics

CaptureEnPort
UpdateEnPort
ShiftEnPort
ScanOutPort
ScanInterface
DataInPort

ShiftEnPort

Syntax

```
Module module_name {
  ShiftEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the shift enable port of a Module's client ScanInterface.

This port is used to enable the shift operation into a module's ScanRegister elements. A module may have zero or more shift enable ports. For an internal module; that is, not a hand-off module, a single shift enable port is implied if there are scan in/out ports in the module. In this case, a ShiftEnPort need not be declared. The ShiftEnPort must be declared if:

- The module is a hand-off module.
- There is more than one shift enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The shift enable signal is an active high signal. A ShiftEnPort of an instance is sourced by a ToShiftEnPort. A ScanRegister object is expected to hold its scan data, as long as the shift enable port is low, independently on the number of clock cycles.

Parameters

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ShiftEnPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name = att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The shift enable port is named "se". Observe that

the body of the Module does not further use the shift enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple shift enable signals.

```
Module tdr2 {

   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

CaptureEnPort
UpdateEnPort
ScanInPort
ScanOutPort
ScanInterface

CaptureEnPort

Syntax

```
Module module_name {
  CaptureEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the capture enable port of a Module's client ScanInterface.

This port is used to control the capture operation of a module's ScanRegister elements. A module may have zero or more capture enable ports. For an internal module; that is, not a hand-off module, a single capture enable port is implied if there are scan in/out ports in the module. In this case, a CaptureEnPort need not be declared. The CaputureEnPort must be declared if:

- The module is a hand-off module.
- There is more than one capture enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The capture enable signal is an active high signal. A CaptureEnPort of an instance is sourced by a ToCaptureEnPort or by a DataOutPort, that in its fanin resolves to a single ToCaptureEnPort. If the source of the capture enable port resolves to 0, the capture enable pulse is assumed to be suppressed.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

Parameters

- *port_name*

  A required string that identifies the name of a capture enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL CaptureEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The capture enable port is named “ce”. Observe that the body of the Module does not further use the capture enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple capture enable signals.

```
Module tdr2 {

   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

ShiftEnPort
UpdateEnPort
ScanInPort
ScanOutPort
ScanInterface

UpdateEnPort

Syntax

```
Module module_name {
  UpdateEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the update enable port of a Module's client ScanInterface.

This port is used to control the update operation of a module's ScanRegister elements. A module may have zero or more capture enable ports. A module may have zero or more update enable ports. For an internal module; that is, not a hand-off module, a single update enable port is implied if there are scan in/out ports in the module. In this case, an UpdateEnPort need not be declared. The UpdateEnPort must be declared if:

- The module is a hand-off module.
- There is more than one update enable port in the ICL module definition. This may happen if there are multiple scan in/out ports defined that are operated independently.

The update enable signal is an active high signal. An UpdateEnPort of an instance is sourced by a ToUpdateEnPort or by a DataOutPort, that in its fanin resolves to a single ToUpdateEnPort. If the source of the update enable port resolves to 0, the update enable pulse is assumed to be suppressed.

Note

The suppressing of the enable signal described above is currently not supported in Tessent.

Parameters

- *port_name*

  A required string that identifies the name of an update enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL UpdateEnPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as

defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows an ICL module definition with a single scan-in / scan-out chain, therefore, no client ScanInterface is required. The update enable port is named “ue”. Observe that the body of the Module does not further use the update enable port name, nor is its active-high value used or stated anywhere. Its connection to the scan chain is done implicitly. Please see the examples of the ScanInterface to learn how you can use multiple scan chains, potentially with multiple update enable signals.

```
Module tdr2 {

   ScanInPort       si;
   ScanOutPort      so  { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
      DefaultLoadValue 8'b0 ;
   }
}
```

Related Topics

ShiftEnPort
CaptureEnPort
ScanInPort
ScanOutPort
ScanInterface

DataInPort

Syntax

```
Module module_name {
  DataInPort port_name {
    RefEnum enum_name ;
    DefaultLoadValue default_load_value;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the data in port of a Module.

A module may have zero or more data in ports. The number and bus-width of all declared data in ports is not correlated to the number and bus-width of any DataOutPort.

A data in port is used to execute a parallel write (that is, not a scan operation), to ports of the module. For a scan operation, use the ScanInPort port function instead. If parallel writes to ports of the module are allowed, the DataInPort function(s) must be declared for the module. If a module does not have DataInPort port functions, no parallel data write is allowed. Certain additional restrictions apply as explained next.

There should be at most one RefEnum per DataInPort, and the widths of the data within the referenced enumeration table should match the width of the data in port.

If a data in port is used to select a scan path, for example, as the select of a ScanMux, it may not be target of a write operation. This ensures that the scan selection remains under control of the software application.

If a data in port is connected to an Instance or DataRegister with an AddressValue statement, all data in ports are concatenated in the order of declaration until the required data width is achieved. See the discussion of AddressValue for further explanation and an example.

Parameters

- *port_name*

  A required string that identifies the name of a data in port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL DataInPort of the module that is also used for the respective ports of the module in the design.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an Enum table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. It must be understood that the enumeration table is not necessarily a

complete list. It is still allowed to write to the data in ports any value, including those not listed in the enumeration table. A DataInPort may have at most one RefEnum statement.

- DefaultLoadValue *default_load_value*

  An optional keyword-string pair that identifies the value that each DataIn port will assume if no user-defined value had been written to it beforehand and other bits of the Ports needs to be written to. If used, the width of *default_load_value* must match the width of the port.

  For DataInPorts on the current design, the tool will automatically force those DataInPorts to the specified *default_load_value* at the beginnign of the pattern. There is one exception to this behavior. If the DataInPort is declared as an inout port in the design module having the same name as the top level ICL module, the port is driven to Z at the beginning of the pattern and will remain Z until the port is the target of an explicit iWrite command or its value is needed to justify other iRead or iWrite commands.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example 1

The following example shows an ICL module definition with only DataInPort, DataOutPort, and a ClockPort port functions. This module does not include any scan interface. Its only way of data IO is through the parallel data port in[7:0] and out[7:0]. Observe that the body of the module is empty, not containing any other ICL objects, like DataRegister, or LogicSignal.

```
Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;
}
```

Example 2

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how the data in port is used inside the ScanRegister. It is declared as the source of the capture values of the scan register, which means that the scan register will capture the value from the data in ports fq on the positive edge of the CaptureEnPort ce, under the condition that the SelectPort en is also high. Finally, observe the usage of the RefEnum and Enum statements.

```
Module tdr_ref {
   ScanInPort       si;
   ScanOutPort      so       {  Source R[0];   }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
```

```
    TCKPort          tck;
    DataInPort       fq[7:0]  { RefEnum OptionalValues; }
    DataOutPort      td[7:0]  { Source R[7:0]; RefEnum OptionalValues; }

    ScanRegister R[7:0] {
       CaptureSource fq[7:0];
       ScanInSource  si;
       RefEnum       OptionalValues ;
    }

    Enum OptionalValues {
        reset = 8'b0 ;
        set   = 8'b11111111 ;
        chain = 8'b11001100 ;
        green = 8'hee ;
        blue  = 8'd198 ;
    }
}
```

Example 3

The following example shows how a data input port can be used to select a scan path. In such a case, the data input port falls under the control of the Tessent, which will determine the necessary data in values to fulfill any read or write requests to the scan chains.

```
Module msel {
  ScanOutPort so { Source smux; }
  DataInPort  dataSel;
  ScanMux smux SelectedBy dataSel {
    1’b0 : tdr0[0];
    1'b1 : tdr1[0];
  }
  [...]
}
```

Related Topics

DataOutPort
ScanInPort
ScanOutPort
Enum

SelectPort

Syntax

```
Module module_name {
  SelectPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the select input port of a Module's client ScanInterface.

A module may have zero or more select ports. It is used to enable a scan interface of a module. When the select port is active (high), the scan interface is active. However, if the select port is inactive (low), the scan interface will not respond to any other control signal defined for this scan interface. The SelectPort statement for a non-handoff module is optional and implied of the module has a scan interface. The SelectPort is only required if the port is part of the ScanInterface statement. A SelectPort which is not part of a scan interface and which is not part of the scan network configuration logic is equivalent to a DataInPort.

Parameters

- *port_name*

  A required string that identifies the name of a select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL SelectPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

In this example, the ShiftEnPort, CaptureEnPort, and UpdateEnPort ports are shared among both ScanInterface. The selection between the two interfaces is accomplished through the two SelectPort sel1 and sel2, respectively.

```
Module chip2 {
   ScanInPort  siA1 ;
   ScanInPort  siA2 ;
   ScanOutPort soA1 { Source blockA.so1 ; }
```

```
      ScanOutPort soA2 { Source blockA.so2 ; }

      ScanInPort  siB1 ;
      ScanInPort  siB2 ;
      ScanOutPort soB1 { Source blockB.so1 ; }
      ScanOutPort soB2 { Source blockB.so2 ; }

      SelectPort   selA ;
      SelectPort   selB ;

      ShiftEnPort   se ;
      CaptureEnPort ce ;
      UpdateEnPort  ue ;

      TCKPort tck ;

      ScanInterface cA12 {
         Port selA;
         Chain c1 { Port siA1; Port soA1 ; }
         Chain c2 { Port siA2; Port soA2 ; }
      }

      ScanInterface cB12 {
         Port selB;
         Chain c1 { Port siB1; Port soB1 ; }
         Chain c2 { Port siB2; Port soB2 ; }
      }
      [...]
   }
```

Related Topics

ScanInPort
DataInPort
ScanInterface
Chain

ResetPort

Syntax

```
Module module_name {
  ResetPort port_name {
    Attribute att_name = att_value ;
    ActivePolarity 0 | 1;
  }
}
```

Description

A port function that declares the name of the reset input port of a Module's client ScanInterface.

You have the option to chose the active polarity of the port. The active polarity is determined for you, if the port is connected to a data port of another instance, or to a TRSTPort.

Parameters

- *port_name*

  A required string that identifies the name of a reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- *ActivePolarity 0|1*

  An optional keyword-value pair, that defines if the reset port of the module is considered active low (value = 0), or active high (value =1). If the ActivePolarity statement is not given, the active polarity of the reset port is assumed to be high.

  The reset port may have an additional data signal in its fanin. (Consider the reset signal of the module being the OR of the reset signal of the parent module and the data signal). In this case the reset port is considered active if either data signal is high (1) or the parents reset signal is active. An additional condition is that there must be a unique reset signal in the parent module.

Note

The functionality described in the last paragraph is called a local reset. It is currently not supported in Tessent.

Related Topics

ScanOutPort

ScanInterface

TMSPort

Syntax

```
Module module_name {
  TMSPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the TMS input port of a Module's client TAP ScanInterface.

If a module has a client TAP interface, the TMSPort declaration may not be omitted. If the module has a ToIRSelectPort, the module must also have exactly one TMSPort.

IEEE P1687 supports top level TAPs, as well as any number of embedded TAP controllers.

Parameters

- *port_name*

  A required string that identifies the name of the TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias

  It is recommended to use a port name for the ICL TMSPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IEEE P1687 has built-in the knowledge of a TAP's state machine.

```
Module tap {
   TCKPort          tck;
   ScanInPort       tdi;
   ScanOutPort      tdo        {  Source IRMux; }
   TMSPort          tms;
   TRSTPort         trst;
   ToSelectPort     tdrEn1     {  Source sel1;  }
```

```
    ScanInPort      fromTdr1;
    ToSelectPort    tapEn1    {  Source IR[2]; }
    ToCaptureEnPort tce;
    ToShiftEnPort   tse;
    ToUpdateEnPort  tue;

    ScanInterface clientTAP {
      Port tdi; Port tdo;
      Port tms; Port trst;
    }

    ScanInterface hostTAP   {
      Port fromTdr1; Port tdrEn1;
      Port tce; Port tse; Port tue;
    }

    Instance fsm_i Of fsm  {
       InputPort tck  = tck;
       InputPort tms  = tms;
       InputPort trst = trst;
     }

    ScanRegister IR[2:0] {
       CaptureSource  3'b001;
       ResetValue     3'b000;
       ScanInSource   tdi;
    }

    ScanRegister bypass {
       CaptureSource  1'b0;
       ScanInSource   tdi;
    }

    ScanMux IRMux SelectedBy fsm_i.irSel {
       1'b0 : DRMux;
       1'b1 : IR[0];
    }

    ScanMux DRMux SelectedBy IR[2:0] {
       3'bx00 : bypass;
       3'bx01 : fromTdr1;
    }

    LogicSignal sel1 {  IR[2:0] == 3'bx01; }
 }

 Module fsm {
    TCKPort        tck;
    TMSPort        tms;
    TRSTPort       trst;
    ToIRSelectPort irSel;
    ToResetPort    tlr;
 }
```

Related Topics

ScanInterface
TRSTPort
ToIRSelectPort

TCKPort

Syntax

```
Module module_name {
  TCKPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the test clock (TCK) input port of a Module.

The declaration of the TCKPort is optional for an internal module (that is, not for a hand-off module), even if the module contains scan registers. In this case, the test clock is implicitly assumed.

A module may have more than one TCKPort statement. In this case, all declared test clocks will be considered equivalent and with the same polarity.

Non test clock ports are defined through the ClockPort statement.

Note that IEEE P1687 has no means of defining the actual speed of a test clock. This is left to the software implementing the standard. In Tessent, you can use the add_clocks command to define clock properties, like the period or the off-value.

Parameters

- *port_name*

  A required string that identifies the name of the test clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotScanGroup, OneHotDataGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL TCKPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name = att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

ScanInterface
ScanRegister
ClockPort

ClockPort

Syntax

```
Module module_name {
  ClockPort port_name {
    Attribute att_name = att_value ;
    DifferentialInvOf other_clock_port_name ;
  }
}
```

Description

A port function that declares the name of a functional clock input port of a Module (that is, a clock that is not a test clock).

A test clock is defined through the TCKPort statement. A module may have zero or more clock ports.

Note that IEEE P1687 has no means of defining the actual speed of a functional clock. This is left to the software implementing the standard. In Tessent, you can use the add_clocks command to define clock properties, like the period, or if the clock is synchronous or asynchronous to the test clock. Inside of ICL, only the ToClockPort statement can be used to model an oscillator-like behavior.

Parameters

- *port_name*

  A required string that identifies the name of a clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ClockPort of the module that is also used for the respective ports of the module in the design.

- DifferentialInvOf *other_clock_port_name*

  An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example declares one test clock and two functional, non-test clocks, an ordinary one and a differential one, the latter of which consists of two clock pins “pdiff” and “ndiff”.

```
Module top {
   TCKPort tck;
   ClockPort sclk;
   ClockPort pdiff;
   ClockPort ndiff { DifferentialInvOf pdiff ; }
}
```

Related Topics

TCKPort

TRSTPort

Syntax

```
Module module_name {
  TRSTPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a TAP reset input port of a Module's client TAP ScanInterface. The active polarity of the port is low.

Especially for embedded TAP controllers, it becomes important to gate-off the global TRST signal. This is accomplished by assuming an AND gate between a data signal and global, unique TRST signal. Note that if the TRST port has a data signal in its fanin in addition to the unique TRST signal, the module's TRST signal is assumed suppressed if the data signal evaluates to 0.

Note

The functionality described in the last paragraph is called a local reset. It is currently not supported in Tessent.

Parameters

- *port_name*

  A required string that identifies the name of the TRST port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL TRSTPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

ResetPort
TMSPort
ScanInterface

AddressPort

Syntax

```
Module module_name {
  AddressPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of an address input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the address of the addressable instances and data registers.

Parameters

- *port_name*

  A required string that identifies the name of an address port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL AddressPort of the module that is also used for the respective ports of the module in the design.

  The source of the address port name signal may only be an address port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  If a module has more than one AddressPort, the address comparison with logic signals of the module implicitly concatenates all address port signals in order of declaration (top to bottom).

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

WriteEnPort
Instance
ReadEnPort
DataRegister

WriteEnPort

Syntax

```
Module module_name {
  WriteEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the write enable input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the write enable to the addressable instance and data registers.

Parameters

- *port_name*

  A required string that identifies the name of a write enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

  The source of the write enable port name signal may only be a write enable port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  A module may have at most one WriteEnPort statement. A module must have a WriteEnPort statement, if the module contains writable, addressable data registers or writable, addressable instances.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

Instance
ReadEnPort
DataRegister

ReadEnPort

Syntax

```
Module module_name {
  ReadEnPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the read enable input port of a Module. It is used if the module contains addressable instances or data registers.

This port serves as the read enable to the addressable instance and data registers.

A read operation is executed in two steps. At first the ReadEnPort must be activated (high) and the address must be supplied (AddressPort). In the second step, the capture is executed and the data is being read. During this second step the ReadEnPort is automatically deactivated (low) by means of an automatic iApplyEndState property (see the Alias statement).

Parameters

- *port_name*

  A required string that identifies the name of a read enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanInPort of the module that is also used for the respective ports of the module in the design.

  The source of the read enable port name signal may only be a read enable port of the parent module, or scan register bits of a ScanRegister statement. In both cases, this connection must be direct, and may not include any logic signals.

  A module may have at most one ReadEnPort statement. A module must have a ReadEnPort statement, if the module contains readable, addressable data registers or readable, addressable instances.

  A read enable port is assumed to have an automatic iApplyEndState property. See the Alias statement to learn more about iApplyEndState.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

AddressPort
Alias
DataRegister
Instance
WriteEnPort

ScanOutPort

Syntax

Syntax 1

```
Module module_name {
  ScanOutPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ScanOutPort port_name {
    Source source_name ;
    Enable enable_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a scan-out port of a Module.

A module may have zero or more scan-out ports. However, if a module has at least one ScanInPort defined, then there must be at least one ScanOutPort.

The scan-out port is used to execute a serial read (that is, not a parallel read operation). For a parallel read, use the DataOutPort port function instead.

If none of the parameters Source, Enable, or Attribute are used, you must use Syntax 1. However, in this case, the scan chain is broken at the ScanOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken scan segment will result in an error during PDL command retargeting.

Parameters

- *port_name*

  A required string that identifies the name of a scan-out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ScanOutPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  A keyword-string pair that identifies the source of the scan-out data, like a ScanRegister or a ScanOutPort of another module instance. A ScanOutPort may have at most one Source statement. The Source statement is not needed for black box modules, which are currently not supported by Tessent. If the source_name references a ScanRegister, it should reference the right most bit of the register. Further, the width of the Source must match the width of the ScanOutPort.

- Enable *enable_name*

  A keyword-string pair that identifies the enable signal of the scan-out port. The enable signal must become active in order to connect the scan-out port with the scan source. The activation and deactivation of the enable signal is taken care of by Tessent. Please see the OneHotScanGroup to learn more about the usage of the enable signal. A ScanOutPort may have at most one Enable statement.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows the usage of the ScanOutPort and its Source statement. Observe that the width of the scan-out port is 2, hence the concatenation of the right most bits of both scan registers. Please see the discussion and examples of the OneHotScanGroup to learn more about the usage of the ScanOutPort's enable signal.

```
Module tdr2x {
   ScanInPort       si[1:0];
   ScanOutPort      so[1:0] { Source R0[0],R1[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanInterface chain2x {
     Port se ;
     Chain c0 { Port si[0]; Port so[0]; }
     Chain c1 { Port si[1]; Port so[1]; }
   }

   ScanRegister R0[9:0] {
      ScanInSource     si[0];
      ResetValue       10'b0 ;
   }

   ScanRegister R1[4:0] {
      ScanInSource     si[1];
      ResetValue       5'b0 ;
   }
}
```

Related Topics

ScanInPort
DataOutPort
ScanInterface
OneHotScanGroup

DataOutPort

Syntax

Syntax 1

```
Module module_name {
  DataOutPort port_name ;
}
```

Syntax 2

```
Module module_name {
  DataOutPort port_name {
    Source source_name ;
    Enable enable_name ;
    RefEnum enum_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the data out port of a Module.

A module may have zero or more data out ports. The number and bus-width of all declared data out ports is not correlated to the number and bus-width of any DataInPort.

A data out port is used to execute a parallel read (that is, not a scan operation) to ports of the module or to readable objects inside the module, like DataRegister. If data is to be scanned-out, use the ScanOutPort port function instead. If a parallel read from ports or objects of the module is to be allowed, the DataOutPort function(s) must be declared for the module. If a module does not have DataOutPort port functions, no parallel data read is allowed. Certain additional restrictions apply as explained next.

If the data output port signal has SelectPort or ToSelectPort, which is part of an ScanInterface in its fanin, then it may not be target of an iWrite or iRead operation. This ensures that the scan selection remains under control of the software application.

If a data out port is connected to an Instance or DataRegister with an AddressValue statement, all data out ports are concatenated in the order of declaration until the required data width is achieved. See the discussion of AddressValue for further explanation and an example.

If none of the parameters Source, Enable, Attribute, or RefEnum are used, you must use Syntax 1. However, in this case, the data connection is broken at the DataOutPort. Retargeting cannot use this port, and if no other solution can be found, reading and writing from the broken data connection will result in an error during PDL command retargeting.

Parameters

- *port_name*

  A required string that identifies the name of a data out port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any

of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

It is recommended to use a port name for the ICL DataOutPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the parallel output data, like a DataRegister or a DataOutPort of another module instance. A DataOutPort may have at most one Source statement. Further, the width of the Source must match the width of the DataOutPort.

- Enable *enable_name*

  An optional keyword-string pair that identifies the enable signal of the data out port. The enable signal must become active in order to connect the data output port with the data source. The activation and deactivation of the enable signal is taken care of by Tessent. Please see the to learn more about the usage of the enable signal. A DataOutPort may have at most one Enable statement.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an Related Topics table. Using the enumeration table, the writing to the data in port can use synonyms listed in the table, instead of bit strings. It must be understood that the enumeration table is not necessarily a complete list. It is still allowed to write to the data in ports any value, including those not listed in the enumeration table. A DataOutPort may have at most one RefEnum statement.

- Attribute *att_name = att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example shows an ICL module definition with both a parallel data input/output and a scan interface (implicitly defined, since there is only one scan-in / scan-out pair). Observe how the data in port is used inside the ScanRegister. It is declared as the source of the capture values of the scan register, which means that the scan register will capture the value from the data in ports fq on the positive edge of the CaptureEnPort ce, under the condition that the SelectPort en is also high. Finally, observe the usage of the RefEnum and Related Topics statements.

```
Module tdr_ref {
   ScanInPort       si;
   ScanOutPort      so      {  Source R[0];   }
   ShiftEnPort      se;
   SelectPort       en;
```

```
        CaptureEnPort    ce;
        UpdateEnPort     ue;
        TCKPort          tck;
        DataInPort       fq[7:0]  { RefEnum OptionalValues; }
        DataOutPort      td[7:0]  { Source R[7:0]; RefEnum OptionalValues; }

        ScanRegister R[7:0] {
           CaptureSource fq[7:0];
           ScanInSource  si;
           RefEnum       OptionalValues ;
        }

        Enum OptionalValues {
            reset = 8'b0 ;
            set   = 8'b11111111 ;
            chain = 8'b11001100 ;
            green = 8'hee ;
            blue  = 8'd198 ;
        }
    }
```

Related Topics

DataInPort
ScanInPort
ScanOutPort
Enum

ToShiftEnPort

Syntax

Syntax 1

```
Module module_name {
  ToShiftEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToShiftEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-shift enable port of a Module's host ScanInterface.

This port is used to enable the shift operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-shift enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToShiftEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-shift enable signal. A ToShiftEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToShiftEnPort.

  A to-shift enable signal pointing to a shift enable signal may evaluate to a constant 0 or a constant 1. If it evaluates to a constant 0, the scan network controlled by the to-shift enable signal is considered disabled. No read or write operation into this scan chain segment may occur. If it evaluates to a constant 1, it is considered an error.

Note

The Source statement is currently not supported.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IEEE P1687 has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;

   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }

   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }

   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }

   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
      ScanInSource   tdi;
   }
```

```
      ScanMux IRMux SelectedBy fsm_i.irSel {
         1'b0 : DRMux;
         1'b1 : IR[0];
      }

      ScanMux DRMux SelectedBy IR[2:0] {
         3'bx00 : bypass;
         3'bx01 : fromTdr1;
      }

      LogicSignal sel1 {  IR[2:0] == 3'bx01; }
   }

   Module fsm {
      TCKPort        tck;
      TMSPort        tms;
      TRSTPort       trst;
      ToIRSelectPort irSel;
      ToResetPort    tlr;
   }
```

Related Topics

TMSPort
ScanInterface
ToIRSelectPort
ToTCKPort
ToSelectPort
ToCaptureEnPort
ToUpdateEnPort
ToResetPort

ToCaptureEnPort

Syntax

Syntax 1

```
Module module_name {
  ToCaptureEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToCaptureEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-capture enable output port of a Module's host ScanInterface.

This port is used to trigger the capture operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToCaptureEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-capture enable signal. A ToCaptureEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToCaptureEnPort.

  The Source may point to a data signal. The to-capture enable signal is considered inactive, if the data signal resolves to 0. Conversely, if the data signal must be resolved to 1 by the tool, to enable the to-capture enable signal. In either case, Tessent will automatically compute the appropriate data signal values for the requested read or write operation to the dependent module's scan network. In addition, if Source points to a data signal, the module may have at most one capture enable signal. You can think of this as an AND gate with one input being the data signal, and the other being the capture enable signal.

Note
The Source statement is currently not supported.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example 1

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IEEE P1687 has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;

   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }

   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }

   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
```

```
         CaptureSource  1'b0;
         ScanInSource   tdi;
      }

      ScanMux IRMux SelectedBy fsm_i.irSel {
         1'b0 : DRMux;
         1'b1 : IR[0];
      }

      ScanMux DRMux SelectedBy IR[2:0] {
         3'bx00 : bypass;
         3'bx01 : fromTdr1;
      }

      LogicSignal sel1 {  IR[2:0] == 3'bx01; }
   }

   Module fsm {
      TCKPort        tck;
      TMSPort        tms;
      TRSTPort       trst;
      ToIRSelectPort irSel;
      ToResetPort    tlr;
   }
```

Related Topics

TMSPort
ScanInterface
ToIRSelectPort
ToTCKPort
ToSelectPort
ToUpdateEnPort
ToShiftEnPort
ToResetPort

ToUpdateEnPort

Syntax

Syntax 1

```
Module module_name {
  ToUpdateEnPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToUpdateEnPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-update enable port of a Module's host ScanInterface.

This port is used to trigger the update operation of another module's ScanRegister elements. It is an active high output signal. A module may have zero or more to-update enable ports.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a shift enable port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToUpdateEnPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-update enable signal. A ToUpdateEnPort may have at most one Source statement. Further, the width of the Source must match the width of the ToUpdateEnPort.

  The Source may point to a data signal. The to-update enable signal is considered inactive, if the data signal resolves to 0. Conversely, if the data signal must be resolved to 1 by the tool, to enable the to-update enable signal. In either case, Tessent will automatically compute the appropriate data signal values for the requested read or write operation to the dependent module's scan network. In addition, if Source points to a data signal, the module may have at most one update enable signal. You can think of this as an AND gate with one input being the data signal, and the other being the update enable signal.

Note
The Source statement is currently not supported.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example 1

The following shows a complete example ICL module definition of a TAP controller. Observe that the state machine is not described, there is only a generic module that defines the instruction register select signal. IEEE P1687 has built-in the knowledge of a TAP's state machine. Also observe the usage of the various To*EnPort statements.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;

   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }
   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }

   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }

   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
```

```
         CaptureSource  1'b0;
         ScanInSource   tdi;
      }

      ScanMux IRMux SelectedBy fsm_i.irSel {
         1'b0 : DRMux;
         1'b1 : IR[0];
      }

      ScanMux DRMux SelectedBy IR[2:0] {
         3'bx00 : bypass;
         3'bx01 : fromTdr1;
      }

      LogicSignal sel1 {  IR[2:0] == 3'bx01; }
   }

   Module fsm {
      TCKPort        tck;
      TMSPort        tms;
      TRSTPort       trst;
      ToIRSelectPort irSel;
      ToResetPort    tlr;
   }
```

Related Topics

TMSPort
ScanInterface
ToIRSelectPort
ToTCKPort
ToSelectPort
ToCaptureEnPort
ToShiftEnPort
ToResetPort

ToResetPort

Syntax

Syntax 1

```
Module module_name {
  ToResetPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToResetPort port_name {
    Attribute att_name = att_value ;
    Source source_name ;
    ActivePolarity 0 | 1;
  }
}
```

Description

A port function that declares the name of the to-reset output port of a Module's host ScanInterface.

The to-reset port is used as the source of the ResetPort of another module. You have the option to chose the active polarity of the to-reset port.

It is important to differentiate between the active polarity, 0 or 1, and when a to-reset signal becomes active / inactive. For example, it is possible that the to-reset signal's active polarity is 0, but it is enabled if the module's input reset port turns active high (1).

If none of the parameters Source, Attribute, or ActivePolarity are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-reset port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotScanGroup, OneHotDataGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToResetPort of the module that is also used for the respective port of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ActivePolarity *0|1*

  An optional keyword-value pair, that defines if the to-reset port of the module is considered active low (value = 0), or active high (value =1). A module may have at most one ActivePolarity statement. If the ActivePolarity statement is not given, the active polarity of the to-reset port is assumed to be the 1 (high).

  The activity/inactivity of the to-reset signal is given through the single ResetPort statement of the module, or, if the module does not have a ResetPort or more than one, through the global reset signal. In other words, if the reset signal is active, the to-reset signal is active as well, however, the ActivePolarity may be different.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-reset signal. A ToResetPort may have at most one Source statement. However, if the module has more than one ResetPort statement, then the Source must be specified. Further, the width of the Source must match the width of the ToResetPort.

  The to-reset signal is assumed to be active, if the source signal evaluates to 1. Note that this does not necessarily mean that the to-reset signal also evaluates to 1, since the active polarity might have been set to 0.

Note

The Source statement is currently not supported.

Example

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. It is an error to connect a ToResetPort to a ResetPort with different polarities.

```
Module chip_reset {
   ScanInPort  si ;
   ScanOutPort so { Source inst_inst.so ; }

   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   SelectPort    selC ;

   ResetPort re { ActivePolarity 1 ; }
   TCKPort tck ;

   Instance res_inst Of res {
      InputPort re   = re ;
   }

   Instance inst_inst Of inst {
      InputPort si  = si ;
      InputPort se  = se;
      InputPort ce  = ce ;
      InputPort ue  = ue ;
```

```
         InputPort sel = selC ;
         InputPort re  = res_inst.trp ;
      }
   }

   Module res {
      ResetPort re ;
      ToResetPort trp { ActivePolarity 0 ; }
   }

   Module inst {
      ScanInPort  si ;
      ScanOutPort so { Source R[0] ; }

      ShiftEnPort   se ;
      CaptureEnPort ce ;
      UpdateEnPort  ue ;
      SelectPort    sel  ;

      TCKPort tck ;

      ResetPort re { ActivePolarity 0; }

      ScanRegister R[7:0] {
         ScanInSource si ;
         ResetValue 8'b0 ;
      }
   }
```

Related Topics

ScanInterface
ToTCKPort
ToUpdateEnPort
ToSelectPort
ToCaptureEnPort
ToShiftEnPort

ToSelectPort

Syntax

Syntax 1

```
Module module_name {
  ToSelectPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToSelectPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-select port of a Module's host ScanInterface.

This port is typically used to enable the scan operation in another module. It is an active high output signal. A module may have none, one, or several to-select ports.

Parameters

- *port_name*

  A required string that identifies the name of a to-select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToSelectPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort.

  An optional keyword-string pair that identifies the source of the to-select signal. The width of the Source must match the width of the ToUpdateEnPort. If the to-select diagonal fans out into a ScanInterface, which is referenced by an *AllowBroadcastingOnScanInterface* property, anywhere in the hierarchy, then the Source statement is mandatory.

Note

Currently Tessent does not support any broadcasting.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the

language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following shows an example ICL netlist that switches the active polarity of the reset signal between instances. Observe that the outgoing active polarity of res.trp is low, so is the incoming polarity of the module instance inst_inst.re. Further, the SelectPort signal for the instance inst_inst is computed based on the top level DataIn signal din.

```
Module chip {
   ScanInPort  si ;
   ScanOutPort so { Source inst_inst.so ; }

   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;

   DataInPort din ;
   ResetPort re { ActivePolarity 1 ; }

   TCKPort tck ;

   Instance res_inst Of res {
      InputPort gres = din ;
      InputPort re   = re ;
   }

   Instance inst_inst Of inst {
      InputPort si  = si ;
      InputPort se  = se;
      InputPort ce  = ce ;
      InputPort ue  = ue ;
      InputPort sel = res_inst.tsel ;
      InputPort re  = res_inst.trp ;
   }
}

Module res {
   ResetPort    re ;
   DataInPort   gres ;
   ToResetPort  trp  { ActivePolarity 0 ; }
   ToSelectPort tsel { Source gres; }
}

Module inst {
   ScanInPort  si ;
   ScanOutPort so { Source R[0] ; }

   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;
   SelectPort    sel  ;
```

```
      TCKPort tck ;

      ResetPort re { ActivePolarity 0; }

      ScanRegister R[7:0] {
         ScanInSource si ;
         ResetValue 8'b0 ;
      }
   }
```

Related Topics

ToShiftEnPort
ToUpdateEnPort
ToResetPort
ToCaptureEnPort

ToClockPort

Syntax

Syntax 1

```
Module module_name {
  ToClockPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToClockPort port_name {
    Source source_name ;
    FreqMultiplier pos_int ;
    FreqDivider pos_int ;
    DifferentialInvOf other_clock_port_name ;
    Period pos_int unit ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of a functional to-clock output port of a Module (that is, a clock that is not a test clock).

The test clock is defined through the ToTCKPort statement. A module may have zero or more to-clock ports.

A ToClockPort serves as the source of a functional clock to another module. It has three principal features:

- Modify an existing incoming functional clock with respect to its frequency by multiplication and division of the period
- Serve as an oscillator clock generation with a defined outgoing clock frequency
- Convert from differential to single-ended clock signals and vice versa.

If none of the parameters are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Source *source_name*

  An optional keyword-string pair that identifies the source of the to-clock signal. A ToClockPort may have at most one Source statement. The width of the Source must match the width of the ToClockPort. The usage of Source is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqMultiplier *pos_int*

  An optional keyword-positive integer number pair that identifies the multiplication factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqMultiplier statement. The usage of FreqMultiplier is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- FreqDivider *pos_int*

  An optional keyword-positive integer number pair that identifies the division factor of the to-clock outgoing clock signal with respect to an incoming clock signal of the module, referenced by Source. A ToClockPort may have at most one FreqDivider statement. The usage of FreqDivider is mutually exclusive with the usage of Period as well as with the usage of DifferentialInvOf within the same ToClockPort statement.

- DifferentialInvOf *other_clock_port_name*

  An optional keyword-string pair. Differential functional clocks are declared through the DifferentialInvOf keyword. The *other_clock_port_name* string references another ClockPort *port_name* of the same module. The usage of DifferentialInvOf is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and Period, leaving the DifferentialInvOf the only parameter other than the Attribute within a ToClockPort statement.

- Period *pos_int unit*

  An optional keyword-positive integer number pair, followed by an optional unit string. If used, the unit string is exactly one of these time units: s, ms, us, ns, ps. If unit is not used, the default is “ns”.

  Using the Period parameter, you can create a module that does not require an incoming clock signal to generate an outgoing clock signal of a specified period. This module is like an oscillator.

  The usage of Period is mutually exclusive from the usage of Source, FreqMultiplier, FreqDivider, and DifferentialInvOf, leaving the Period as the only parameter other than the Attribute within a ToClockPort statement.

Note

Currently Tessent does not support the Period parameter.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example represents a programmable clock generation logic that selectively uses one of two incoming reference clocks and manipulates either with a selectable set of modifiers.

```
Module chip_clock {
   ScanInPort    si ;
   ScanOutPort   so { Source clockGen_inst.so ; }
   ShiftEnPort   se ;
   CaptureEnPort ce ;
   UpdateEnPort  ue ;

   TCKPort   tck ;
   ClockPort refClkA ;
   ClockPort refClkB ;

   DataInPort  din[7:0] ;
   DataOutPort dout[7:0] { Source inst_inst.out ; }

   Instance clockGen_inst Of clockGen {
      InputPort si = si ;
      InputPort refClkA = refClkA ;
      InputPort refClkB = refClkB ;

      Parameter sclkMult = 1 ;
      Parameter sclkDiv  = 3 ;
   }

   Instance inst_inst Of instrument {
      InputPort clk = clockGen_inst.sclk ;
      InputPort in  = din ;
   }
}

Module clockGen {
   ScanInPort    si ;
   ScanOutPort   so { Source clkSelReg[0] ; }

   ClockPort refClkA ;
   ClockPort refClkB ;

   Parameter sclkMult = 1 ;
   Parameter sclkDiv  = 1 ;

   ToClockPort sclk {
     FreqMultiplier $sclkMult ;
     FreqDivider $sclkDiv ;
```

```
      Source refSelM ;
   }

   ScanRegister clkSelReg[2:0] {
      ScanInSource si ;
      ResetValue 3'b100 ;
   }

   ClockMux refSelM SelectedBy clkSelReg[2] {
      1'b1 : refClkA ;
      1'b0 : refClkB ;
   }
}

Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;

   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
   Alias go          = out[0]          { RefEnum PassFail; }
   Alias done        = out[1]          { RefEnum YesNo;    }
   Alias count[5:0]  = out[7:2];

   Enum PassFail {
      Pass = 1'b1;
      Fail = 1'b0;
   }
   Enum YesNo {
      Yes  = 1'b1;
      No   = 1'b0;
   }
   Enum Modes {
      red   = 4'b0011;
      blue  = 4'b1000;
      green = 4'b0100;
   }
}
```

Related Topics

ClockPort

TCKPort
ToTCKPort

ToTCKPort

Syntax

Syntax 1

```
Module module_name {
  ToTCKPort port_name ;
}
```

Syntax

```
Module module_name {
  ToTCKPort port_name {
    Attribute att_name = att_value ;
}
```

Description

A port function that declares the name of a test to-test clock output port of a Module (that is, a clock that is not a functional clock).

The test clock is defined through the ToClockPort statement. A module may have zero or more to-test clock ports. If a module has several to-test clock outputs, all clocks are considered equivalent in period and polarity.

Unlike the ToClockPort statement, there are no modifiers in the ToTCKPort statement. The outgoing to-test clock signal is just a pass-through of the incoming test clock.

If no parameter is used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-clock port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToClockPort of the module that is also used for the respective ports of the module in the design.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Example

The following example modifies the example given in ToClockPort. Here, a TCKPort and a ToTCKPort port was added to the module. Observe that ToTCKPort is just passing throught he test clock. No manipulation of the test clock signal can happen.

```
Module clockGen {
   ScanInPort    si ;
   ScanOutPort   so { Source clkSelReg[0] ; }

   TCKPort   tck ;
   ClockPort refClkA ;
   ClockPort refClkB ;

   Parameter sclkMult = 1 ;
   Parameter sclkDiv  = 1 ;

   ToTCKPort   ttck ;

   ToClockPort sclk {
     FreqMultiplier $sclkMult ;
     FreqDivider $sclkDiv ;
     Source refSelM ;
   }

   ScanRegister clkSelReg[2:0] {
      ScanInSource si ;
      ResetValue 3'b100 ;
   }

   ClockMux refSelM SelectedBy clkSelReg[2] {
      1'b1 : refClkA ;
      1'b0 : refClkB ;
   }
}
```

Related Topics

ClockPort

TCKPort
ToClockPort

ToTMSPort

Syntax

Syntax 1

```
Module module_name {
  ToTMSPort port_name ;
  }
}
```

Syntax 2

```
Module module_name {
  ToTMSPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-TMS port of a Module's host TAP ScanInterface. This port is typically used to provide a TMS signal to an embedded TAP controller module or to module that contains an embedded TAP controller.

If Source is used, the outgoing to-TMS signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToTMSPort of the module that is also used for the respective ports of the design module.

  Any module that has a ToTMSPort must also have at least one TMSPort.

- Source *source_name*

  An optional keyword-string pair that identifies the logic signal that is considered gating the to-TMS signal of the module. A ToTMSPort may have at most one Source statement. Source is required if the module has more than one TMSPort. If the module has only one TMSPort, Source is optional and if not provided, ToTMSPort then refers to the TMSPort.

  If Source points to a data signal, the module must have exactly one TMSPort. The outgoing to-TMS signal is low (0) when the data signal evaluates to 0. If the data signal evaluates to 1, the ToTMSPort follows the TMSPort. In this sense the ToTMSPort is considered AND-gated with the TMS signal and the data signal at the input sides of the AND.

Note
The Source statement is currently not supported.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort
ToTCKPort
ToTRSTPort

ToTRSTPort

Syntax

Syntax 1

```
Module module_name {
  ToTRSTPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToTRSTPort port_name {
    Source source_name ;
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-TRST port of a Module's host TAP ScanInterface. This port is used to distribute the TRST signal throughout the design. The to-TRST outgoing signal is always active low (0).

A module may have zero or more to-TRST ports. If a module has several to-TRST outputs, all to-TRST signals are considered equivalent.

If Source is used, the outgoing to-TRST signal is considered gated.

If neither of the parameters Source or Attribute are used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-TMS port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToTRSTPort of the module that is also used for the respective ports of the design module.

- Source *source_name*

  An optional keyword-string pair that identifies the logic signal that is considered gating the to-TRST signal of the module. A ToTRSTPort may have at most one Source statement, and the width of the Source must match the width of the ToTRSTPort. Source is required if the module has more than one TRSTPort. If the module has only one TRSTPort, Source is optional and if not provided, ToTRSTPort then refers to the TRSTPort. Further, if the module does not have a TRSTPort, the ToTRSTPort refers to the global TRST signal.

  If Source points to a data signal, the module should have at most one TRSTPort. The outgoing to-TRST signal is inactive; that is, high (1), when the data signal evaluates to 0. If the data signal evaluates to 1, the ToTRSTPort follows the TRSTPort. In this sense the

ToTRSTPort is considered AND-gated with the (local or global) TRST signal and the data signal at the input sides of the AND.

Note

The Source statement is currently not supported.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort
ToTCKPort
ToTMSPort

ToIRSelectPort

Syntax

Syntax 1

```
Module module_name {
  ToIRSelectPort port_name ;
}
```

Syntax 2

```
Module module_name {
  ToIRSelectPort port_name {
    Attribute att_name = att_value ;
  }
}
```

Description

A port function that declares the name of the to-IR Select port of a Module.

This signal is used to select between the instruction register of a TAP controller and another scan register, usually a data register. The to-IR Select signal is always associated with the TMS signal, and if present, with the TRST signal.

If the parameter Attribute is not used, you must use Syntax 1.

Parameters

- *port_name*

  A required string that identifies the name of a to-IR Select port of the module. The port names for all ports of a module must be unique. In addition, the port name may not be used for any of the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  It is recommended to use a port name for the ICL ToIRSelectPort of the module that is also used for the respective ports of the design module.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

Related Topics

TMSPort

TRSTPort

Instance

Syntax

Syntax 1

```
Module module_name {
  Instance instance_name Of module_name {
    Attribute att_name = att_value ;
    InputPort moduleinputport_name = concat_signal ;
    Parameter parameter_name = parameter_value ;
    AddressValue number ;
  }
}
```

Syntax 2

```
Module module_name {
  Instance instance_name Of module_name ;
}
```

Description

The Instance statement names an instance of an ICL module. It also declares the address and/or all input connections of the instance to source signals of other instances or the top level module. Note that in ICL no output connections are listed in the Instance statement.

In addition to the input port connections, Parameters defined within the module definition may be overwritten. LocalParameter cannot be overwritten though.

Syntax 2 must be used if the module has no input ports and none of the other parameters is used.

Parameters

- *instance_name* Of *module_name*

  A required string-string pair, connected with the case-sensitive keyword ‘Of’, defining the name of the instance and referencing an existing module name of which an instance is to be created. Within a module, all instance names must be unique. Also, no module may contain an instance of itself anywhere through its hierarchy.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value are defined by you, and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- InputPort *moduleinputport_name* = *concat_signal*

  A repeatable, required keyword and string-value pair, connected with the ‘=’ symbol. For each input port defined in the module there must be exactly one such line in the Instance statement, however exceptions apply as explained below. *moduleinputport_name* must refer to an input port name of the referenced module. For modules without any input ports

the InputPort parameter may not be used. The order of the input port declaration in the referenced module and the order of the InputPort parameters can be different.

concat_signal may reference a single signal or a concatenation of signals of the same or compatible type. The type of *concat_signal* must match the type of the input port definition in the module, or must be a compatible type. *concat_signal* may only be one of these signals: reset, scan, data, read enable, write enable, clock, TCK, shift enable, capture enable, update enable, TMS, or TRST. The width of the module's port definition and *concat_signal* must match. For an indexed input port of the module, the InputPort parameter may be used as often as necessary to cover the entire index range in any order. However, overlapping indices or ranges are not allowed. For instances without the AddressValue parameter, the InputPort parameter may connect the module's WriteEnPort, ReadEnPort, AddressPort, or SelectPort to data signals; that is, a data signal is a compatible source for those port functions.

Note that not all input ports of a module must be connected explicitly. For a non-handoff module; that is, for a module that is not the top level module you hand over to a third party, several connections are automatically implied, if their source in the parent module can be determined unambiguously. It is legal to list these connections through the InputPort parameter, but it is not necessary. The following connections are automatically implied for non-hand-off module instances that do not have the AddressValue statement: TCK, shift enable, capture enable, update enable.

- Parameter *parameter_name* = *parameter_value*

  A repeatable, optional keyword and string-value pair, connected with the "=" symbol. The referenced *parameter_name* must have been defined in the module referenced by *module_name*; also the type of *parameter_value* must match the type used in the module for the parameter.

  Using the Parameter statement again within Instance overwrites the respective parameter value set in the module definition, but only for the current instance. Another instance of the module may use another overwrite value, or may use the original value defined in the module. Through this construct, ICL can define parameterized modules, where the final parameter is defined during each instantiation of the module.

- AddressValue *number*

  A required keyword and number pair, if this instance is part of the OneHotDataGroup. The AddressValue parameter may not be used if the instance if not part of a OneHotDataGroup. Within the OneHotDataGroup, the *number* declares a part of the address that is associated with the instance. The final address of a DataRegister is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense *number* is only one element of the sum, comprising the address of a data register of the OneHotDataGroup.

  The following connections are automatically implied for non-hand-off module instances that do have the AddressValue statement: TCK, the instance's AddressPort is assumed to be connected to the logical address signal of the parent module, the instance's WriteEnPort is assumed to be connected to the WriteEnPort of the parent module, the instance's

ReadEnPort is assumed to be connected to the ReadEnPort of the parent module, and the instance's DataInPort is assumed to be connected to the logical data in signals of the parent module.

Note

In IEEE P1687 the Instance statement describes the "AllowBroadcastOnScanInterface" parameter. This parameter is currently not supported in Tessent.

Example 1

The following example demonstrates the usage of automatic default connections in the common case of a scan interface. Observe that the instance declaration for tdr_inst of module tdr only declares the scan input port connection. TCK and all the enable signals are connected fully automatically to the respective signals of the parent module. Explicit connections are only necessary, if there are multiple signals of the same type in the parent module, for example two select ports.

```
Module chip_tdr {
   ScanInPort       si;
   ScanOutPort      so       { Source tdr_inst.so; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   Instance tdr_inst Of tdr {
     InputPort si = si ;
   }
}

Module tdr {
   ScanInPort       si;
   ScanOutPort      so       { Source R[0]; }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   ScanRegister R[7:0] {
      ScanInSource     si;
      ResetValue       8'b0 ;
   }
}
```

Example 2

The following example demonstrates not only the usage of the OneHotDataGroup, but also the usage of the InputPort parameter and some of the compatible signal connections. Observe that module "top" does have neither a ReadEnPort, nor a WriteEnPort. Instead it has 2 DataInPort statements that are connected to the respective input ports of the instances of module "phy". Since the connection of the data signals of the parent module (that is, top) to the instances is

done implicitly, by the order of declaration, it is important that the 16 data input bits of DataIn are declared before the 2 bits of the write and read enable. Also note that the module top defines just one AddressPort of 8 bits, which is implicitly mapped to the two 4-bit AddressPorts of phy in their order of declaration. Further, observe that the AddressPort of “bank” only 2-bits wide, since only two bits are required for this module’s operation.

```
Module top {
   AddressPort   address[7:0] ;
   DataInPort    dataIn[15:0] ;

   DataInPort    write_enable ;
   DataInPort    read_enable ;

   DataOutPort   dataOut[15:0] {
     Source outMux[15:0] ;
     Enable read_enable ;
   }

   OneHotDataGroup outMux[15:0] {
     Instance phy1 Of phy {
       AddressValue 2'b01, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }

     Instance phy2 Of phy {
       AddressValue 2'b10, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }
   }
}

Module phy {
   WriteEnPort 7 we ;
   ReadEnPort    re ;
   AddressPort   address1[3:0] ;
   AddressPort   address2[3:0] ;

   DataInPort    dataIn1[7:0] ;
   DataInPort    dataIn2[7:0] ;
   DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }

   OneHotDataGroup outMux[15:0] {
     Instance bank1 Of bank {
       AddressValue 4'b0001,2'b00 ;
     }
     Instance bank2 Of bank {
       AddressValue 4'b0010,2'b00 ;
     }
   }
}

Module bank {
   WriteEnPort   we ;
   ReadEnPort    re ;
   AddressPort   address[1:0];
```

```
      DataInPort    dataIn[15:0];
      DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }

      OneHotDataGroup outMux[15:0] {
//      AddressValue 2'bx,4'bx,2'b<num>
        DataRegister D0[15:0] { AddressValue 2'b00 ; }
        DataRegister D1[15:0] { AddressValue 2'b01 ; }
        DataRegister D2[15:0] { AddressValue 2'b10 ; }
        DataRegister D3[15:0] { AddressValue 2'b11 ; }
      }
    }
```

Related Topics

ScanInterface
DataRegister
ScanRegister
OneHotDataGroup
OneHotScanGroup

Parameter

Syntax

Syntax 1 : Defining a parameter

```
Module module_name {
  Parameter parameter_name = parameter_value ;
}
```

Syntax 2 : Optionally overwriting a parameter in an Instance statement

```
Module module_name {
  Instance instance_name of other_module_name {
    Parameter parameter_name = parameter_value ;
  }
}
```

Description

With the help of the Parameter statement you can define a named variable and assign a default value to it.

The default value is specified in the module that declares the Parameter. The value can be overwritten with every instance of the Module. Through this, ICL allows you to create parameterized modules.

You build a reference to the Parameter by using the '$' symbol in front of the parameter_name.

The parameter reference can be used in many locations, for example in an Enum statement. The reference will be resolved to the current value before the statement the Parameter is evaluated.

Parameters

- *parameter_name = parameter_value*

  A required string-value pair, connected with the '=' symbol. The parameter name must be unique across all Parameter and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an Instance name "inst$size", where "size" is a parameter

  Although parameters can be used to define other parameters, a circular definition is not allowed.

Note

Currently Tessent does not support the Callback functionality.

Example 1

The following example shows several usages of Parameter. In particular observe that the Parameter ‘Width’ is used to parameterize the values in the enumeration table.

```
Module inst {
   ScanInPort       si;
   ScanOutPort      so      {  Source R[0];   }
   TCKPort          tck;

   Parameter  Width  = 8 ;
   Parameter  Regmax = $Width - 1 ;
   Parameter  userValue = 8'hff ;
   Parameter  Val_A  = 4'b1001, $userValue, 4'b1111 ;
   Parameter  RegName = "R" ;

   LocalParameter  Regmin = 0 ;

   ScanRegister R[$Regmax:$Regmin] {
      Attribute     Register = "$RegName($RegMax:$RegMin)" ;
      ScanInSource  si;
      ResetValue    Reset ;
      RefEnum       SomeValues ;
   }

   Enum SomeValues {
       Reset = $Width'b0 ;
       User1 = $userValue ;
   }
}
```

Example 2

The following example uses the module defined above. In its instantiation, the parameters ‘Width’ and ‘userValue’ will be overwritten. This allows ICL to create parameterized modules, where the size of registers will be fixed during the instantiation of the module and not during the declaration of the module.

```
Module top {
   ScanInPort       si;
   ScanOutPort      so      {  Source inst2.so;   }
   ShiftEnPort      se;
   SelectPort       en;
   CaptureEnPort    ce;
   UpdateEnPort     ue;
   TCKPort          tck;

   Instance inst1 Of inst {
      InputPort si = si ;
      Parameter Width = 16 ;
      Parameter userValue = 16'hffff ;
   }

   Instance inst2 Of inst {
      InputPort si = inst1.so ;
      Parameter Width = 4 ;
      Parameter userValue = 4'ha ;
```

```
        }
    }
```

Related Topics

LocalParameter
Instance
Alias
Enum

LocalParameter

Syntax

```
Module module_name {
  LocalParameter parameter_name = parameter_value ;
}
```

Description

A LocalParameter has the same semantic and use model as the Parameter statement. The only difference is that a LocalParameter cannot be overwritten in the Instance statement.

Parameters

- *parameter_name = parameter_value*

  A required string-value pair, connected with the '=' symbol. The parameter name must be unique across all Parameter and LocalParameter statements within the module. The parameter value may be exactly one of the following: an unsized number, a sized number, or a string. A parameter reference representing an unsized number may only be used where an unsized number is expected. Similarly for the sized number. A parameter reference representing a string may only be used in an attribute within any statement in the module, or within a Callback function. Places where a parameter reference is allowed include an attribute value, the index value for a vector identifier, or where a size or unsized number is allowed. For example, you cannot use a parameter to substitute any arbitrary number, such as creating object names, like an Instance name "inst$size", where "size" is a parameter

  Although parameters can be used to define other parameters, a circular definition is not allowed.

Related Topics

Parameter
Instance
Alias
Enum

Enum

Syntax

```
Module module_name {
  Enum enum_name {
    name = value ;
  }
}
```

Description

With the Enum statement, ICL introduces an enumeration table that links a mnemonic string to an arbitrary value. This string can then be used in lieu of the value at places like DataInPort, DataOutPort, and LogicSignal. For the ScanRegister and the DataRegister the string can be used for the DefaultLoadValue, ResetValue, and CaptureSource (ScanRegister only).

Further, PDL commands like iWrite or iRead can also use the name instead of the value. Choosing meaningful mnemonics can make both, ICL and PDL, much more human readable.

In no usage of the enumeration table are you limited to only the names listed in the table. You are free to continue using any legal value, and in particular values not listed in the table. The example below shows such a case.

Parameters

- *enum_name*

  A required string that identifies the name of the enumeration table. Using this name the RefEnum parameter inside the ScanRegister statement, for example, can reference the enumeration table. The *enum_name* must be unique among all Enum statements within the module.

- *name = value*

  A required, repeatable string-value pair, connected with the '=' symbol. These pairs make up the enumeration table. All values must be of the same width, and the names must be unique within the same Enum. An enumeration table must contain at least one name-value pair.

Example

The following enumeration table shows several examples of value definitions. The width of the value depends on the usage of the table entries in other objects, like ScanRegister or DataInPort, referenced by RefEnum. within these statements. The number type is not relevant as shown. Also observe that two Parameter were defined, which are used in the enumeration table 'Modes'. As usual, these parameters could be overwritten differently in each instance of the module.

```
Module instrument {
   DataInPort  in[7:0];
   DataOutPort out[7:0];
   ClockPort   clk;
```

```
   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
   Alias go          = out[0]          { RefEnum PassFail; }
   Alias done        = out[1]          { RefEnum YesNo;    }
   Alias count[5:0]  = out[7:2];

   Parameter userEnum1val = 4'b1111 ;
   Parameter userEnum2val = 4'h22 ;

   Enum PassFail {
      Pass = 1'b1;
      Fail = 1'b0;
   }

   Enum YesNo {
      Yes  = 1'b1;
      No   = 1'b0;
   }

   Enum Modes {
      none      = 4'b0 ;
      red       = 4'b0011 ;
      blue      = 4'haa ;
      green     = 4'd9 ;
      userEnum1 = $userEnum1val ;
      userEnum2 = $userEnum2val ;
   }
}
```

Next, here is a snippet of a dofile, writing to the ‘mode’ Alias of the module, using the names defined in the ‘Mode’ enumeration table which is associated with the alias through the RefEnum parameter. The done Alias is then read from ‘out’.

```
open_pattern_set enumTest
  iWrite mode red
  iApply
  iWrite mode 0b1010
  iApply
  iRead  done Yes
  iApply
close_pattern_set
write_patterns enumTest.pdl -pdl
```

The following lists the contents of the saved PDL file enumTest.pdl:

```
iProcsForModule instrument

iProc enumTest {} {

   iWrite in[6] 0b0
   iWrite in[5] 0b0
   iWrite in[3] 0b1
   iWrite in[2] 0b1
   iApply

   iWrite in[6] 0b1
   iWrite in[5] 0b0
```

```
        iWrite in[3] 0b1
        iWrite in[2] 0b0
        iApply

        iRead out[1] 0b1
        iApply
    }
```

Related Topics

iRead
LogicSignal
ScanRegister
iWrite
DataInPort
DataOutPort
DataRegister

Alias

Syntax

Syntax 1

```
Module module_name {
  Alias alias_name = concat_signal ;
}
```

Syntax 2

```
Module module_name {
  Alias alias_name = concat_signal {
    RefEnum enum_name ;
    AccessTogether ;
    iApplyEndState value ;
  }
}
```

Description

The Alias statement collects signals from different origins to form a named, new, virtual signal.

The name of the alias can be used in any operation or construct where the original signals would have been allowed. For example, PDL commands like iWrite or iRead can use the alias name. Choosing meaningful alias names can make both ICL and PDL, much more human readable.

A second usage of the Alias statement is to instruct Tessent to automatically return to a specified value for the selected signals. Typically this feature is used to return a write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

If none of the parameters RefEnum, AccessTogether, or iApplyEndState are used, you must use Syntax 1.

The name of the Alias must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal.

Parameters

- *alias_name = concat_signal*

  A required string-string pair, connected with the '=' symbol. The *alias_name* defines the name of the alias, whereas *concat_signal* refers to one signal, or a concatenation of signals. Such a concatenation of signals is created by a signal name followed by a comma (','), followed by another signal name etc. The concatenated signals can only be from DataInPort, ScanInPort, DataOutPort, or ScanOutPort. The width of the alias name should match the width of the concatenated signal.

  An alias name can become part of another Alias statement, however circular definitions are not allowed. Further, if any of the signals in the concatenation has more than one level of hierarchy; that is, the hierarchical instance-port path name contains the dot ("."), the Alias may not be used in any other ICL statements, but only in PDL.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an Related Topics table. Using the enumeration table, the writing to the aliased signal can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. You are still allowed to read or write any value, including those not listed in the enumeration table. An Alias statement may have at most one RefEnum parameter.

- AccessTogether

  An optional keyword that requests Tessent to operate all signals listed in *concat_signal* at the same time. If this turns out to be not possible, for example, based on mutually exclusive hardware requirements, an error will be returned. Without this keyword, Tessent is allowed to find a solution where some of the signals are operated first, followed by operating another set of signals in the *concat_signal* Alias.

Note

Currently Tessent does not support the AccessTogether parameter.

- iApplyEndState *value*

  An optional keyword-string pair that identifies the value to which the signals listed *concat_signal* will return to automatically, if ever changed by an iWrite command. The value will return with the next iApply PDL command. Typically this feature is used to return the write-enable signal to the inactive value assignment automatically, once the write operation has been completed.

  There are additional conditions to the signals and value used in the Alias and iApplyEndState statement and parameter: All resolved signals must be legal targets of the iWrite command; *value* may not contain unknown values; if a signal is part of multiple iApplyEndState parameters, all specified values must be the same; if the signals resolve to register bits, either ScanRegister or DataRegister, the register must have a ResetValue parameter defined and the reset value should match the iApplyEndState-value for these signals; if a signal does not resolve to register bits, it must resolve to primary inputs, no other resolved signal source is allowed for the *concat_signal*, if the iApplyEndState parameter is used; further, there may not be a DataOutPort with an Enable parameter between the resolved signal source and the signal specified in *concat_signal* of the Alias statement.

Example

The following example shows several usages of the Alias statement.

```
Module instrument {
   DataInPort  in[8:0];
   DataOutPort out[7:0];
   ClockPort   clk;

   Alias write_en    = in[8]{ RefEnum YesNo; iApplyEndState 1'b0; }
   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes;    }
   Alias p1[2:0]     = in[4],in[1:0];
```

```
      Alias go          = out[0]          { RefEnum PassFail; }
      Alias done        = out[1]          { RefEnum YesNo;    }
      Alias count[5:0]  = out[7:2];

      Enum PassFail {
         Pass = 1'b1;
         Fail = 1'b0;
      }

      Enum YesNo {
         Yes  = 1'b1;
         No   = 1'b0;
      }

      Enum Modes {
         none      = 4'b0 ;
         red       = 4'b0011 ;
         blue      = 4'haa ;
         green     = 4'd9 ;
      }
   }
```

Related Topics

iRead
LogicSignal
ScanRegister
iWrite
DataInPort
DataOutPort
DataRegister

LogicSignal

Syntax

```
Module module_name {
  LogicSignal signal_name {
    expression ;
  }
}
```

Description

The LogicSignal statement creates a new signal named *signal_name*, which itself can then be referenced in other ICL statements, and in particular other LogicSignal statements.

Using the LogicSignal statement, logical operations between signals like AND or EXOR can be implemented.

Parameters

- *signal_name*

  A required string that identifies the newly created signal name. The value it assumes depends on the result of the evaluation of *expression*.

  The name of the LogicSignal must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, Alias.

  In any case, the width of signal_name is always 1. In particular, comparisons and Boolean logic operations return 1’b1, if the expression evaluates to true, and 1’b0, if the expression evaluates to false.

- *expression*

  A required string that defines a Boolean expression that evaluates to a single-bit result, based on the signals and their value used in the expression.

  The common Boolean and Logical operators are supported. In decreasing order of preference, these are: Boolean not (!), Bitwise not (~), logical AND (&&) and logical OR(||), bitwise AND (&), bitwise OR (|), and bitwise XOR (^), logical equality (==) and ineqeuality (!=).

  All operators are left-to-right associative, and operators with equal precedence are evaluated left-to-right. Associativity and precedence may be expressly specified with parentheses ().

  Valid operands are registers, ports, sized and unsized numbers, logical signals, as well as parameters and aliases that resolve into those. In addition, signals can be concatenated.

  For expressions containing unsized numbers, it must be possible to determine the required width of the unsized number unambiguously. If the width must be extended, the additional bits will either be filled with 0, if the most significant bit is either 0 or 1, or the bits will be filled with unknown (x).

For the logical equality and inequality operation, the width of the operands on either side of the operator must be the same.

Any parameter reference will be resolved before the expression is evaluated.

Enumeration table entries can be used for operands which are used in their entirety in the expression, and not just a part of the operand. The operand must have declared its binding to this enumeration by means of the RefEnum statement. Enumeration values are allowed only in combination with equality (==) and inequality (!=) operators. See the example below.

Example

The following example shows several usages of the LogicalSignal statement. sig1 demonstrates the usage of an enumeration table value entry. Note that ‘mode[3:0]’ is the entire operand for which the Enum ‘Modes’ applies. Therefore, any table entry may be used for this operand. sig2 shows the usage of concatenation, the expansion of an unsigned number and the usage of a parameter reference. Observe that the width of the right operand is 5, whereas the width of the sized elements of the left operand is 3. Therefore, the unsigned number ‘0’ will be expanded to width 2, matching the width of the right operand. Lastly, sig3 shows an example of bit-operations.

```
Module instrument {
   [ ... ]
   DataInPort  in[7:0];
   [ ... ]
   Alias enable      = in[7]           { RefEnum YesNo;    }
   Alias mode[3:0]   = in[6:5],in[3:2] { RefEnum Modes; }

   Parameter user1 = 4'b1010 ;

   LogicSignal sig1 { (mode==red) && (enable==1'b1) ; }
   LogicSignal sig2 {  in[4],1'b1,in[0],0 == 1'b0,$user1 ; }
   LogicSignal sig3 { (in[6:5] | in[3:2]) != 2'b10 ; }

   Enum Modes {
      none      = 4'b0 ;
      red       = 4'b0011 ;
      blue      = 4'haa ;
      green     = 4'd9 ;
   }
   [ ... ]
}
```

ScanRegister

Syntax

```
Module module_name {
  ScanRegister register_name {
    Attribute att_name = att_value;
    ScanInSource scan_in_source;
    CaptureSource capture_source;
    DefaultLoadValue default_load_value;
    ResetValue reset_value;
    RefEnum enum_name;
  }
}
```

Description

The statement ScanRegister defines and names a scan register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the scan register.

Parameters

- *port_name*

  A required string that identifies the name of a the scan register of the module. The name of the ScanRegister must be unique among all port names, and among all the following objects: DataRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

- Attribute *att_name* = *att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- ScanInSource *scan_in_source*

  A required keyword-string pair that identifies the signal that is providing the scan data to the left most bit of the scan chain. If *scan_in_source* references another scan register, it must reference the right-most bit of that scan register (that is, the ‘scan out bit’).

  For ranged scan registers; that is, scan registers that have a width > 1, the scan register always shifts from left to right, independently if the left most index is larger than the right most index, or vice-versa.

- CaptureSource *capture_source*

  An optional keyword-string pair that identifies the signal from which the ScanRegister captures data in response to the CaptureEnPort and TCKPort signal. If used, the width of *capture_source* must match the width of the register.

- DefaultLoadValue *default_load_value*

  An optional keyword-string pair that identifies the value that each scan register cell will assume if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- ResetValue *reset_value*

  An optional keyword-string pair that identifies the value that each scan register cell will assume as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the scan register does not respond to the module's incoming reset signal. However, if the module contains scan registers with a ResetValue parameter, the module must have only one ResetPort statement.

  When both the DefaultLoadValue parameter the ResetValue parameter exist in the same Scan Register statement, the non-x bit values of *reset_value* mast match those of *default_load_value*.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an Related Topics table. Using the enumeration table, reading and writing to the scan register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A ScanRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the scan register.

Additional Rules

A ScanRegister is considered to be in the active scan path if it responds to the ShiftEnPort signal associated to the ScanInterface through an active ScanInterface. It holds its data as long as the ShiftEnPort remains inactive (low) between the capture- and the update-cycle.

Any scan register bit that controls parts of a scan network (that is, the select line of a ScanMux, a SelectPort or ToSelectPort), or is part of a ScanInterface, anywhere in the hierarchy of the design, will have a non-x reset value (that is, a reset value of 0 or 1).

If a scan register bit is the source of a data signal, the scan register must have an update stage that (or an equivalent implementation) which changes only in response to an active UpdateEnPort signal.

It is important to understand the rules which determine what value gets loaded into the scan register bits. Assume no user-defined care bits have ever been written to any scan register bits. Now an iApply block accesses the ScanRegister. The scan register bits will assume a value in this order of preference: The specified values required to satisfy the operations in the iApply

block; the value DefaultLoadValue specifies, if is a 0 or 1; the value ResetValue specifies, if it is a 0 or 1; any scan register bits still unassigned a 0 or 1, will be loaded with a 0.

If a scan register receives a reset signal, all scan register bits assume the value specified in *reset_value*. For the remaining x-values apply the sequence of events outlined in the paragraph above.

If a ScanRegister is on the active scan path (that is, a shift operation may happen), any scan register bits that are not required to satisfy the operations in the current iApply block will remain unchanged from their current value.

Example 1

The following shows a complete example ICL module definition of a TAP controller. Observe that the two different ScanRegisters: 'IR' has a width of 3, whereas 'bypass' has a width of 1. 'IR' must have as ResetValue parameter defined, since it controls the ScanMux DRMux. Further, observe the usage of the name of the scan register in the expression of the LogicSignal 'sel1'.

```
Module tap {
   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToSelectPort    tapEn1    {  Source IR[2]; }
   ToCaptureEnPort tce;
   ToShiftEnPort   tse;
   ToUpdateEnPort  tue;

   ScanInterface clientTAP {
     Port tdi; Port tdo;
     Port tms; Port trst;
   }

   ScanInterface hostTAP   {
     Port fromTdr1; Port tdrEn1;
     Port tce; Port tse; Port tue;
   }

   Instance fsm_i Of fsm  {
      InputPort tck  = tck;
      InputPort tms  = tms;
      InputPort trst = trst;
    }

   ScanRegister IR[2:0] {
      CaptureSource  3'b001;
      ResetValue     3'b000;
      ScanInSource   tdi;
   }

   ScanRegister bypass {
      CaptureSource  1'b0;
```

```
      ScanInSource   tdi;
   }

   ScanMux IRMux SelectedBy fsm_i.irSel {
      1'b0 : DRMux;
      1'b1 : IR[0];
   }

   ScanMux DRMux SelectedBy IR[2:0] {
      3'bx00 : bypass;
      3'bx01 : fromTdr1;
   }

   LogicSignal sel1 {  IR[2:0] == 3'bx01; }
}

Module fsm {
   TCKPort        tck;
   TMSPort        tms;
   TRSTPort       trst;
   ToIRSelectPort irSel;
   ToResetPort    tlr;
}
```

Example 2

The following example shows an ICL module definition of a sensor where CaptureSource can be used to combine multiple DataInPorts to form the width of the ScanRegister.

```
Module sensor {
   TCKPort        tck;
   TRSTPort       rst;
   ScanInPort     si;
   ScanOutPort    so;      {  Source  S_reg[0]; }
   CaptureEnPort  capture_en;
   ShiftEnPort    shift_en;
   UpdateEnPort   update_en;
   SelectPort     sel_en;
   DataInPort     sdata_in [2:0];
   DataInPort     sens_from;
   DataInPort     cntrl [1:0];

ScanRegister S_reg [5:0] {
   ResetValue     6’b000000;
   CaptureSource  sdata_in, sens_from, cntrl ;
   ScanInSource   si;
  }
}
```

Related Topics

ScanInPort
DataOutPort
iApply
DataRegister
ShiftEnPort

DataRegister

Syntax

Syntax 1

```
Module module_name {
  DataRegister register_name {
    WriteDataSource concat_signal ;
    WriteEnSource enable_source ;

    ResetValue reset_value ;
    DefaultLoadValue load_value ;
    RefEnum enum_name ;
    Attribute att_name = att_value;
  }
}
```

Syntax 2

```
Module module_name {
  DataRegister register_name {
    AddressValue address_value ;

    ResetValue reset_value ;
    DefaultLoadValue load_value ;
    RefEnum enum_name ;
    Attribute att_name = att_value;
  }
}
```

Description

The statement DataRegister defines and names a data register of a certain width and with certain properties. These properties, like the reset value, are described through parameters of the data register. The DataRegister is used for parallel reading and writing as opposed to a ScanRegister, which offers a scan-based access.

There are three, mutually exclusive types of DataRegisters: A Selectable DataRegister, for which a signal must be defined from which the DataRegister captures data; an Addressable DataRegister, for which an AddressValue must be specified; and a CallBack DataRegister. You should use Syntax 1 if you have a Selectable DataRegister. You should use Syntax 2 if you have an Addressable DataRegister.

Further, the Addressable DataRegister can only be used inside a OneHotDataGroup, and inside a OneHotDataGroup there can only be Addressable DataRegisters.

Note

Currently Tessent does not support the Callback functionality.
Also the DefaultLoadValue parameter is not supported at the moment.

Parameters

- *register_name*

  A required string that identifies the name of a data register of the module. The name of the DataRegister must be unique among all port names, and among all the following objects: ScanRegister, ScanMux, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

- ResetValue *reset_value*

  An optional keyword-string pair that identifies the value that each data register cell will assume as a response to an active reset signal of the module. In more detail, it is the update stage of the scan register that assumes the declared reset value. If used, the width of *reset_value* must match the width of the register. If the ResetValue parameter is not used, the data register does not respond to the module's incoming reset signal. However, if the module contains data registers with a ResetValue parameter, the module must have only one ResetPort statement.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

Note

Currently Tessent supports only the reset of a DataRegister with a ResetValue parameter following the global reset signal.

- DefaultLoadValue *default_load_value*

  An optional keyword-string pair that identifies the value that each data register cell will assume if no user-defined value had been written to it beforehand and if no reset had been triggered. If used, the width of *default_load_value* must match the width of the register.

  When both the DefaultLoadValue parameter and the ResetValue parameter exist in the same DataRegister statement, the non-x bit values of *reset_value* must match those of *default_load_value*.

- RefEnum *enum_name*

  An optional keyword-string pair that references the name of an Enum table. Using the enumeration table, reading and writing to the data register can use synonyms listed in the table, instead of bit strings. Understand that the enumeration table is not necessarily a complete list. It is still allowed to read or write any value, including those not listed in the enumeration table. A DataRegister statement may have at most one RefEnum parameter. In addition, the width of the elements in the enumeration table must match the width of the data register.

- Attribute *att_name = att_value*

  A repeatable, optional keyword, string and value combination. The standard does not have any built-in attributes, only the structure of the Attribute parameter is defined by the language. The attribute name and the attribute value is defined by you and may be ignored

by any IJTAG software tool. Tessent understands several port attributes and pin attributes as defined in this manual. Note that most attributes are read-only and are automatically generated by the tool.

- WriteDataSource *concat_signal*

  A keyword and string pair required for a selectable DataRegister. *concat_signal* references a single signal or a concatenation of signals from where the DataRegister will capture data in response to a write request when WriteEnSource turns active (high). concat_signal must refer either to the module's DataInPorts, another DataRegister, or to a ScanRegister.

- WriteEnSource *enable_source*

  A keyword and string pair required for a selectable DataRegister. *enable_source* references a 1-bit wide signal which determines when the DataRegister can capture data from WriteDataSource. The DataRegister will capture data in response to a write request when WriteEnSource turns active (high).

- AddressValue *address_value*

  A keyword and number pair required if an Addressable DataRegister; that is, the DataRegister is part of the OneHotDataGroup. The AddressValue parameter may not be used if the DataRegister if not part of a OneHotDataGroup. Within the OneHotDataGroup, the *number* declares a part of the address that is associated with the DataRegister. The final address of a DataRegister is determined by the sum of all addresses along the hierarchical instance path leading to the data register. In this sense *number* is only one element of the sum, comprising the address of a data register of the OneHotDataGroup.

  The DataRegister will be loaded from the logical data-in signal with the right most indices first, when the DataRegister's address has been applied and the module's WriteEnPort is active (high). The address of a DataRegister is computed by adding all address_values of all Instances leading to the DataRegister.

  Similarly, reading from the Addressable DataRegister requires the module's ReadEnPort being active (high), and the address of the DataRegister being applied.

Example

The following example shows a selectable data register. An example of addressable data registers can be found with the OneHotDataGroup statement, as well as with the Instance statement.

Observe the usage of the Parameter statement as well as Parameter overwrites. Further observe how the write enable signal is computed for the data registers in module bank using a LogicSignal statement and concatenation of the parameterized signals of the comparisons.

All together, this example shows how a selectable DataRegister can be used very much in a way of an addressable DataRegister, but without any of the constraints that come with an addressable DataRegister. In this particular implementation, the read and write enable signal of bank are explicitly managed (see for example, the usage of Dx_Mux). Depending on your DataRegister implementation, you may be able to relax this and allow reading from and writing

to a data register simultaneously. Tessent sets up constraints for the IJTAG software such that no race conditions can occur.

```
Module chip_selectable_dataregister {
   DataInPort  write_enable ;
   DataInPort  read_enable ;

   DataInPort  address[7:0] ;
   DataInPort  dataIn[15:0] ;
   DataOutPort dataOut[15:0] { Source outMux[15:0] ; }

   Parameter bank1Address = 4'b0011 ;
   Parameter bank2Address = 4'b1100 ;

   DataMux outMux[15:0] SelectedBy address[7:4] {
     $bank1Address : bank1.dataOut[15:0] ;
     $bank2Address : bank2.dataOut[15:0] ;
   }

   Instance bank1 Of bank {
     InputPort we = write_enable ;
     InputPort re = read_enable ;
     InputPort dataIn[15:0] = dataIn[15:0] ;
     InputPort address[3:0] = address[3:0] ;
   }

   Instance bank2 Of bank {
     InputPort we = write_enable ;
     InputPort re = read_enable ;
     InputPort dataIn[15:0] = dataIn[15:0] ;
     InputPort address[3:0] = address[3:0] ;
     Parameter D0_Address = 4'b1000 ;
     Parameter D1_Address = 4'b1100 ;
   }
}

Module bank {
   DataInPort    we ;  Alias weR = we { iApplyEndState 1'b0 ; }
   DataInPort    re ;

   DataInPort    address[3:0];
   DataInPort    dataIn[15:0];
   DataOutPort   dataOut[15:0] { Source Dx_Mux[15:0] ; Enable re; }

   Parameter D0_Address = 4'b0000 ;
   Parameter D1_Address = 4'b0001 ;

   DataMux Dx_Mux[15:0] SelectedBy we,address[3:0] {
     1'b0, $D0_Address : D0[15:0] ;
     1'b0, $D1_Address : D1[15:0] ;
   }

   LogicSignal D0_wen {  re,we,address[3:0] == 2'b01, $D0_Address ; }
   DataRegister D0[15:0] {
     ResetValue       16'b0 ;
     WriteEnSource    D0_wen ;
     WriteDataSource  dataIn[15:0] ;
```

```
        }

        LogicSignal D1_wen {  re,we,address[3:0] == 2'b01, $D1_Address ; }
        DataRegister D1[15:0] {
          ResetValue       16'b0 ;
          WriteEnSource    D1_wen ;
          WriteDataSource  dataIn[15:0] ;
        }
    }
```

Related Topics

DataInPort
DataOutPort
ScanRegister
OneHotDataGroup

ScanMux

Syntax

```
Module module_name {
  ScanMux mux_name SelectedBy mux_select_signal {
    mux_select_value : scan_source_name ;
  }
}
```

Description

The ScanMux statement creates and names a scan multiplexor.

IEEE P1687 differentiates between a scan multiplexor, a data multiplexor, and a clock multiplexor. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexor only switches between scan signals, the data multiplexor only switches between data signals, and the clock multiplexor only switches between clock signals. No signal types may be mixed for any of the multiplexors. The select signal of the multiplexors may be sourced only by data signals. Note that the parallel output of a scan register is also a data signal.

A common design structure is to have a ScanMux, with a 1-bit TDR at the output of the ScanMux, and the parallel output of the TDR connected to the select input of the mux. In this configuration, the TDR behind the mux is always on the scan path, independent of which scan input ports of the mux have been selected. Through this TDR, Tessent can select either of the scan input ports of the mux as needed.

Parameters

- *mux_name*

  A string that identifies the name of a scan multiplexor of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, DataMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the outgoing signal of the multiplexor can be referenced. The width of *mux_name* must match the with of the *scan_source_name*.

- *mux_select_value* : *scan_source_name*

  A repeatable string-string pair, connected by a colon symbol ‘:’. A ScanMux must have at least one *mux_select_value* - *scan_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

  The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *scan_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Example

The following shows usages of all the multiplexor types.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }

   ShiftEnPort   se1 ;
   CaptureEnPort ce1 ;
   UpdateEnPort  ue1 ;

   ShiftEnPort   se2 ;
   CaptureEnPort ce2 ;
   UpdateEnPort  ue2 ;

   TCKPort tck ;
   ClockPort clkA ;
   ClockPort clkB ;
   ToClockPort clk { Source clockM ; }

   DataInPort   dselA ;
   DataInPort   dselB[2:0] ;
   DataOutPort do[3:0] { Source dataM[3:0] ; }

   ScanInterface cA12 {
      Port se1; Port ce1; Port ue1;
      Chain c1 { Port siA; Port so ; }
   }

   ScanInterface cB12 {
      Port se2; Port ce2; Port ue2;
      Chain c1 { Port siB; Port so ; }
   }

   ScanRegister regA[7:0] {
      ScanInSource siA ;
      ResetValue   8'b0 ;
   }

   ScanRegister regB[7:0] {
      ScanInSource siB ;
      ResetValue   8'b0 ;
   }

   ScanMux scanM SelectedBy dselA {
      1'b0 : regA[0] ;
      1'b1 : regB[0] ;
   }

   ClockMux clockM SelectedBy dselA {
      1'b0 : clkA ;
      1'b1 : clkB ;
   }

   DataMux dataM[3:0] SelectedBy dselB[2:0] {
      3'bx00 : regA[7:4] ;
```

```
            3'bx01 : regA[3:0] ;
            3'bx1x : regB[5:2] ;
        }
    }
```

Related Topics

DataMux

ClockMux

DataMux

Syntax

```
Module module_name {
  DataMux mux_name SelectedBy mux_select_signal {
    mux_select_value : data_source_name ;
  }
}
```

Description

The DataMux statement creates and names a data multiplexor.

IEEE P1687 differentiates between a scan multiplexor, a data multiplexor, and a clock multiplexor. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexor only switches between scan signals, the data multiplexor only switches between data signals, and the clock multiplexor only switches between clock signals. No signal types may be mixed for any of the multiplexors. The select signal of the multiplexors may be sourced only by data signals.

Parameters

- *mux_name*

  A string that identifies the name of a data multiplexor of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, ClockMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the outgoing signal of the multiplexor can be referenced. The width of *mux_name* must match the with of the *data_source_name*.

- *mux_select_value* : *data_source_name*

  A repeatable string-string pair, connected by a colon symbol ':'. A DataMux must have at least one *mux_select_value* - *data_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

  The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *data_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Example

The following shows usages of all the multiplexor types.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }

   ShiftEnPort   se1 ;
```

```
      CaptureEnPort ce1 ;
      UpdateEnPort  ue1 ;

      ShiftEnPort   se2 ;
      CaptureEnPort ce2 ;
      UpdateEnPort  ue2 ;

      TCKPort tck ;
      ClockPort clkA ;
      ClockPort clkB ;
      ToClockPort clk { Source clockM ; }

      DataInPort   dselA ;
      DataInPort   dselB[2:0] ;
      DataOutPort do[3:0] { Source dataM[3:0] ; }

      ScanInterface cA12 {
         Port se1; Port ce1; Port ue1;
         Chain c1 { Port siA; Port so ; }
      }

      ScanInterface cB12 {
         Port se2; Port ce2; Port ue2;
         Chain c1 { Port siB; Port so ; }
      }

      ScanRegister regA[7:0] {
         ScanInSource siA ;
         ResetValue   8'b0 ;
      }

      ScanRegister regB[7:0] {
         ScanInSource siB ;
         ResetValue   8'b0 ;
      }

      ScanMux scanM SelectedBy dselA {
         1'b0 : regA[0] ;
         1'b1 : regB[0] ;
      }

      ClockMux clockM SelectedBy dselA {
         1'b0 : clkA ;
         1'b1 : clkB ;
      }

      DataMux dataM[3:0] SelectedBy dselB[2:0] {
         3'bx00 : regA[7:4] ;
         3'bx01 : regA[3:0] ;
         3'bx1x : regB[5:2] ;
      }
   }
```

Related Topics

ScanMux ClockMux

ClockMux

Syntax

```
Module module_name {
  ClockMux mux_name SelectedBy mux_select_signal {
    mux_select_value : clock_source_name ;
  }
}
```

Description

The ClockMux statement creates and names a clock multiplexor.

IEEE P1687 differentiates between a scan multiplexor, a data multiplexor, and a clock multiplexor. All three share the same syntax (other than the keyword), but their semantics differ. The scan multiplexor only switches between scan signals, the data multiplexor only switches between data signals, and the clock multiplexor only switches between clock signals. No signal types may be mixed for any of the multiplexors. The select signal of the multiplexors may be sourced only by data signals.

Parameters

- *mux_name*

  A string that identifies the name of a data multiplexor of the module. The name of the ScanMux must be unique among all port names, and among all the following objects: ScanRegister, DataRegister, ScanMux, DataMux, OneHotDataGroup, OneHotScanGroup, LogicSignal, Alias.

  Using this name, the outgoing signal of the multiplexor can be referenced. The width of *mux_name* must match the with of the *clock_source_name*.

- *mux_select_value* : *clock_source_name*

  A repeatable string-string pair, connected by a colon symbol ':'. A DataMux must have at least one *mux_select_value* - *clock_source_name* pair. The width of the *mux_select_value* must match the width of the *mux_select_signal*.

  The *mux_select_value* describes the value the select signal (*mux_select_signal*) of the mux must assume for the *mux_name* signal to connect to the *clock_source_name* signal. The *mux_select_value* may contain x-values. The collection of all specified *mux_select_value* parameters need not cover the entire number space. Any select value not listed causes the *mux_name* signal to assume x-values of the correct width.

Example

The following shows usages of all the multiplexor types.

```
Module chip2 {
   ScanInPort  siA ;
   ScanInPort  siB ;
   ScanOutPort so      { Source scanM ; }

   ShiftEnPort   se1 ;
```

```
      CaptureEnPort ce1 ;
      UpdateEnPort  ue1 ;

      ShiftEnPort   se2 ;
      CaptureEnPort ce2 ;
      UpdateEnPort  ue2 ;

      TCKPort tck ;
      ClockPort clkA ;
      ClockPort clkB ;
      ToClockPort clk { Source clockM ; }

      DataInPort   dselA ;
      DataInPort   dselB[2:0] ;
      DataOutPort do[3:0] { Source dataM[3:0] ; }

      ScanInterface cA12 {
         Port se1; Port ce1; Port ue1;
         Chain c1 { Port siA; Port so ; }
      }

      ScanInterface cB12 {
         Port se2; Port ce2; Port ue2;
         Chain c1 { Port siB; Port so ; }
      }

      ScanRegister regA[7:0] {
         ScanInSource siA ;
         ResetValue   8'b0 ;
      }

      ScanRegister regB[7:0] {
         ScanInSource siB ;
         ResetValue   8'b0 ;
      }

      ScanMux scanM SelectedBy dselA {
         1'b0 : regA[0] ;
         1'b1 : regB[0] ;
      }

      ClockMux clockM SelectedBy dselA {
         1'b0 : clkA ;
         1'b1 : clkB ;
      }

      DataMux dataM[3:0] SelectedBy dselB[2:0] {
         3'bx00 : regA[7:4] ;
         3'bx01 : regA[3:0] ;
         3'bx1x : regB[5:2] ;
      }
   }
```

Related Topics

ScanMux DataMux

OneHotDataGroup

Syntax

```
Module module_name {
  OneHotDataGroup onehotdatagroup_name {
    Instance ;
    DataRegister ;
    Port concat_signal ;
  }
}
```

Description

The OneHotDataGroup statement implements a convenient way of declaring a named, hierarchically distributed data multiplexor. You would use a OneHotDataGroup if various inputs to the multiplexor are sourced by data registers distributed across multiple instances. This statement simplifies the declaration of such a data multiplexor by automatically making the correct data input and data output connections, including read, write, and address connections.

A OneHotDataGroup may use any combination of the three options declaring the source input signal of the multiplexor. The order of the declarations is not relevant, since each source should have an enable signal defined in such a way that at most one source is selected at a time. The OneHotDataGroup then assumes the value of this selected, unique source, or the x-value, if none of the sources is selected, or if multiple data sources are selected simultaneously.

Parameters

- *onehotdatagroup_name*

  A required string, declaring the name of the OneHotDataGroup. This name can be considered as the name of the distributed multiplexor. As usual, *onehotdatagroup_name* can be used to refer to the output signal of this multiplexor, for example as a source of a DataOutPort statement. The width of *onehotdatagroup_name* must be larger or equal to the width of the connected output ports of the referenced Instance or DataRegister. *onehotdatagroup_name* assumes the value of the one selected source, or the x-value, if none of its sources are enabled.

- Instance

  An optional, repeatable Instance statement. This instance must use the AddressValue parameter.

- DataRegister

  An optional, repeatable DataRegister statement. This DataRegister must use the AddressValue parameter.

- Port *concat_signal*

  An optional, repeatable keyword-string pair. *concat_signal* may reference a data output single signal or a concatenation of such signals. *concat_signal* may also be a hierarchical instance pathname ending at a data source signal. The referenced signal will be considered an input port of the multiplexor represented by the OneHotDataGroup statement. For the

enable signal required for each source of the OneHotDataGroup's source signal, one of the following rules will apply:

- For a referenced DataOutPort, the Enable statement is included in the DataOutPort declaration.
- For a referenced DataOutPort, the Source parameter references another data signal, which has an enabling condition.
- The instance referenced in the instance pathname in a hierarchical *concat_signal* has an AddressValue parameter.
- A primary input port of the top level module is assumed to have an enabling condition from the outside.

Example

The following shows an example of a OneHotDataGroup. Observe how the address of a DataRegister is determined. For example the DataRegister with the hierarchical name phy1.bank1.D1 has the address of 8'b01 000000 + 6'b0001 00 + 2'b01 = 8'b01 0001 01. Similarly, the DataRegister phy2.R has the address 8'b10 000000 + 6'b0010 00 = 8'b10 0010 00.

The Example 2 of the Instance statement shows a variation of this example.

```
Module tap {Module top {
   AddressPort   address[7:0] ;
   DataInPort    dataIn[15:0] ;

   DataInPort    write_enable ;
   DataInPort    read_enable ;

   DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }

   OneHotDataGroup outMux[15:0] {
     Instance phy1 Of phy {
       AddressValue 2'b01, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }

     Instance phy2 Of phy {
       AddressValue 2'b10, 6'b000000 ;
       InputPort we = write_enable ;
       InputPort re = read_enable ;
     }
   }
}

Module phy {
   DataInPort    dataIn1[7:0] ;
   DataInPort    dataIn2[7:0] ;

   WriteEnPort   we ;
   ReadEnPort    re ;
   AddressPort   address1[3:0] ;
```

```
      AddressPort   address2[3:0] ;

      DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }

      OneHotDataGroup outMux[15:0] {

        DataRegister R[15:0] {
          AddressValue 4'b0010,2'b00 ;
        }

        Instance bank1 Of bank {
          AddressValue 4'b0001,2'b00 ;
        }
      }
   }

   Module bank {
      DataInPort    dataIn[15:0];

      WriteEnPort   we ;
      ReadEnPort    re ;
      AddressPort   address[1:0];

      DataOutPort   dataOut[15:0] { Source outMux[15:0] ; }

      OneHotDataGroup outMux[15:0] {
//    AddressValue 2'bx,4'bx,2'b<num>
        DataRegister D0[15:0] { AddressValue 2'b00 ; }
        DataRegister D1[15:0] { AddressValue 2'b01 ; }
        DataRegister D2[15:0] { AddressValue 2'b10 ; }
        DataRegister D3[15:0] { AddressValue 2'b11 ; }
      }
   }
```

Related Topics

Instance
DataRegister
OneHotScanGroup

OneHotScanGroup

Syntax

```
Module module_name {
  OneHotScanGroup onehotscangroup_name {
    Port source_name ;
  }
}
```

Description

The OneHotScanGroup statement implements a convenient way of declaring a named, hierarchically distributed scan multiplexor. You would use a OneHotScanGroup if various inputs to the multiplexor are sourced by scan-out ports distributed across multiple instances. This statement simplifies the declaration of such a scan multiplexor by automatically making the correct scan input and scan output connections.

Parameters

- *onehotscangroup_name*

  A required string, declaring the name of the OneHotScanGroup. This name can be considered as the name of the distributed multiplexor. As usual, *onehotscangroup_name* can be used to refer to the output signal of this multiplexor, for example as a source of a ScanOutPort statement. The width of *source_name* must be equal to the width of *onehotdatagroup_name*. Each scan source must have an enable signal defined in such a way that at most one source is selected at a time when the OneHotScanGroup is in the active scan path. This enable signal is not necessarily directly at the ScanOutPort of *source_name*, but might be located deeper in the hierarchy, for example, through cascaded OneHotScanGroups. When selected, the *onehotdatagroup_name* then assumes the value of this selected source, or the x-value, if none of the sources is selected, or multiple sources would had been selected.

- Port *source_name*

  A required, repeatable keyword-string pair. *source_name* may reference a scan output port single signal or a concatenation of such signals. *source_name* may also be a hierarchical instance pathname ending at a scan output port signal. The referenced scan output port signals will be considered an input port of the multiplexor represented by the OneHotScanGroup statement.

Example

The following shows a example of a OneHotScanGroup. Observe how easy it is to declare the hierarchically distributed scan multiplexor “outScanMux” and the usage of the OneHotScanGroup’s name in the Source statement of the top level scan-out port.

```
Module chip_onehotscangroup {
   ScanInPort       si;
   ScanOutPort      so       { Source outScanMux; }
   ShiftEnPort      se;
   SelectPort       en[3:0];
```

```
      CaptureEnPort    ce;
      UpdateEnPort     ue;
      TCKPort          tck;

      OneHotScanGroup outScanMux {
        Port tdr_0.so ;
        Port tdr_1.so ;
        Port tdr_2.so ;
        Port tdr_3.so ;
      }

      Instance tdr_0 Of tdr {
        InputPort si = si ;
        InputPort en = en[0];
      }
      Instance tdr_1 Of tdr {
        InputPort si = si ;
        InputPort en = en[1];
      }
      Instance tdr_2 Of tdr {
        InputPort si = si ;
        InputPort en = en[2];
      }
      Instance tdr_3 Of tdr {
        InputPort si = si ;
        InputPort en = en[3];
      }
   }

   Module tdr {

      ScanInPort       si;
      ScanOutPort      so       { Source R[0]; Enable en; }
      ShiftEnPort      se;
      SelectPort       en;
      CaptureEnPort    ce;
      UpdateEnPort     ue;
      TCKPort          tck;

      ScanRegister R[7:0] {
         ScanInSource     si;
         ResetValue       8'b0 ;
      }
   }
```

Related Topics

Instance
OneHotDataGroup
ScanRegister

AccessLink

Syntax

```
Module module_name {
  AccessLink instance_name Of STD_1149_1_2001 {
    BSDLEntity entity_name;
    intruction_name {
      ScanInterface { scan_interface_name; ...} //mandatory
      ActiveSignals { signal_name; ...} //optional
    }
  }
}
```

Description

A wrapper that can be used inside the top level ICL Module to link an ICL network to an 1149.1 TAP controller that is described in a BSDL file. The tool currently only supports the BSDL format that is part of IEEE1149.1-2001.

When reading an ICL file which contains an ICL module with an AccessLink Of STD_1149_1_2001, the read_icl command must be used with the -bsdl_file option to provide the path to the referenced BSDLEntity. An error is generated if the specified BSDLEntity is not found in the specified BSDL file.

Parameters

- *instance_name*

  A string used to define the instance name of the AccessLink. The string must start with a letter and only contain letters, numbers and underscores. As shown in the Example below, the *instance_name* is used when refering to the defined *signal_name* in an ICL connection. When the AccessLink is read in the tool, a TAP ICL module is created from the BSDL and instantiated in the top level ICL module in place of the AccessLink using the same *instance_name*. This is also illustrated in the Example below.

- BSDLEntity entity_name;

  A required keyword-string pair which defines the name of the BSDL entity that is referenced by the AccessLink wrapper. The file pointed to by the “read_icl -bsdl_file” option must contain an “Entity *entity_name* is” clause.

- *instruction_name*

  A required string that references one of the instruction names defined in the INSTRUCTION_OPCODE attribute of the BSDL file. When the specified instruction_name is loaded in the INSTRUCTION register, the scan_interface_names listed in the ScanInterface wrapper are placed along the active DR scan path between the TAP_SCAN_IN and TAP_SCAN_OUT ports.

- scan_interface_name

  A repeatable string that references the ScanInterface of ICL module instances. The format of the string is a *leaf_instance_name* followed by an optional “.*scan_interface_name*”

string. The "*.scan_interface_name*" string is optional when the ICL module associated with the *leaf_instance_name* only has one ScanInterface. If the ICL module associated to the *leaf_instance_name* has more than one ScanInterface wrappers, the "*.scan_interface_name*" string is mandatory to identify one of them.

The *scan_interface_name* are serially daisy chained with the first one closer to TDI and the last one closer to TDO.

- signal_name

  A repeatable string that defines signal_names that are high when the *instruction_name* is loaded into the INSTRUCTION register. When a signal_name is not specified inside a given *instruction_name* wrapper, the signal is low when the *instruction_name* is loaded into the INSTRUCTION register. The signal_names are identified as "*instance_name.signal_name*" when refrenced to make connections in the ICL. This is illustrated in the Example below.

Example

Below is an example of a top level ICL module containing an AccessLink. It references a BSDL file which contains the element shown below. After the elaboration which is triggered by the set_current_design command, you can see that a TAP controller was extracted from the BSDL file and instantiated into the top level ICL module. The TAP ports were also extracted and added to the ICL module. The original AccessLink wrapper is commented out. The ClockPorts and their original connections are left unaffected.

Note

If an ICL module with an AccessLink statement has been loaded into memory by means of read_icl, then the commands write_icl and report_icl_modules do not result in usable ICL of this module before it has been subject to set_current_design.

Following is the content of mydesign.icl with an AccessLink wrapper. Notice how the ActiveSignal long_setup is referenced as "bsdl_tap.long_setup" when used to connect the "selectWIR" input port of the "core" instance.

```
Module mydesign {
  ClockPort CLK;
  AccessLink bsdl_tap Of STD_1149_1_2001 {
    BSDLEntity acme4523;
    BP0_LONG_SETUP {
      ScanInterface {core.ijtag0;}
      ActiveSignals {long_setup;}
    }
    BP0_SHORT_SETUP {
      ScanInterface {core.ijtag0;}
    }
    BP1_LONG_SETUP {
      ScanInterface {core.ijtag1;}
      ActiveSignals {long_setup;}
    }
    BP1_SHORT_SETUP {
      ScanInterface {core.ijtag1;}
```

```
      }
    }

    Instance core Of core {
      InputPort selectWIR = bsdl_tap.long_setup;
      InputPort clk = CLK;
    }

  }
```

Following is the section from the BSDL file “acme4523.bsdl” that is used to extract the TAP controller.

```
entity acme4523 is
   attribute TAP_SCAN_RESET of TRST : signal is true;
   attribute TAP_SCAN_IN    of TDI  : signal is true;
   attribute TAP_SCAN_MODE  of TMS  : signal is true;
   attribute TAP_SCAN_OUT   of TDO  : signal is true;
   attribute TAP_SCAN_CLOCK of TCK  : signal is (1.0e+07, BOTH);

attribute INSTRUCTION_OPCODE of TOP: entity is
      "IDCODE           (111111111111111110)," &
      "BYPASS           (111111111111111111)," &
      "EXTEST           (111111111111101000)," &
      "SAMPLE           (111111111111111000)," &
      "PRELOAD          (111111111111111000)," &
      "CLAMP            (111111111111101111) " &
      "BP0_LONG_SETUP   (XXXXXX000111XXX010)," &
      "BP1_LONG_SETUP   (XXXXXX001101XXX010)," &
      "BP2_LONG_SETUP   (XXXXXX001001XXX010)," &
      "BP3_LONG_SETUP   (XXXXXX000101XXX010)," &
      "BP4_LONG_SETUP   (XXXXXX000001XXX010)," &
      "BP0_SHORT_SETUP  (XXXXXX110110XXX010)," &
      "BP1_SHORT_SETUP  (XXXXXX111100XXX010)," &
      "BP2_SHORT_SETUP  (XXXXXX111000XXX010)," &
      "BP3_SHORT_SETUP  (XXXXXX110100XXX010)," &
      "BP4_SHORT_SETUP  (XXXXXX110000XXX010) " ;

end acme4523;
```

Below is a dofile that loads the ICL modules in memory and elaborates them. The modified top level ICL module as well as the created TAP ICL modules are then reported. Notice how the TAP ICL module is called <bsdl_entity_name>_bsdl_tap. Its leaf instance name reused the one that was used by the AccessLink.

```
> set_context patterns -ijtag
> read_icl mydesign.icl -bsdl_file acme4523.bsdl
> read_icl core.icl
> set_current_design
> report_icl_module -module mydesign
Module mydesign {
  ClockPort CLK;
  ScanInPort TDI;
  TMSPort TMS;
  TCKPort TCK;
  TRSTPort TRST;
```

```
    ScanOutPort TDO {
      Source bsdl_tap.tdo;
    }
  /*
    AccessLink bsdl_tap Of STD_1149_1_2001 {
      BSDLEntity acme4523;
      BP0_LONG_SETUP {
        ScanInterface {core.ijtag0;}
        ActiveSignals {long_setup;}
      }
      BP0_SHORT_SETUP {
        ScanInterface {core.ijtag0;}
      }
      BP1_LONG_SETUP {
        ScanInterface {core.ijtag1;}
        ActiveSignals {long_setup;}
      }
      BP1_SHORT_SETUP {
        ScanInterface {core.ijtag1;}
      }
    }
    */

    ScanMux tessent_access_link_mux0 SelectedBy
                  bsdl_tap.BP0_LONG_SETUP_select,
                  bsdl_tap.BP0_SHORT_SETUP_select,
                  bsdl_tap.BP1_LONG_SETUP_select,
                  bsdl_tap.BP1_SHORT_SETUP_select {
      4'b1xxx | 4'bx1xx | 4'bxx1x | 4'bxxx1 : TDI;
     }
    Instance core Of core {
      InputPort selectWIR = bsdl_tap.long_setup;
      InputPort clk = CLK;
      InputPort si = tessent_access_link_mux0;
    }
    Instance bsdl_tap Of acme4523_bsdl_tap {
      InputPort tdi = TDI;
      InputPort tms = TMS;
      InputPort tck = TCK;
      InputPort trst = TRST;
      InputPort BP0_LONG_SETUP_from_scan_out = core.fso0;
      InputPort BP0_SHORT_SETUP_from_scan_out = core.fso0;
      InputPort BP1_LONG_SETUP_from_scan_out = core.fso1;
      InputPort BP1_SHORT_SETUP_from_scan_out = core.fso1;
    }
  }

  > report_icl_module -module acme4523_bsdl_tap
  Module acme4523_bsdl_tap {
    TCKPort tck;
    TMSPort tms;
    TRSTPort trst;
    ScanInPort tdi;
    ScanOutPort tdo {
      Source IRMux;
    }
    ToResetPort testLogicReset {
      ActivePolarity 1;
```

```
  Source fsm.tlr;
}
ToResetPort testLogicResetInv {
  ActivePolarity 0;
  Source ~fsm.tlr;
}
ToCaptureEnPort BP0_LONG_SETUP_capture_en;
ToShiftEnPort BP0_LONG_SETUP_shift_en;
ToUpdateEnPort BP0_LONG_SETUP_update_en;
ToSelectPort BP0_LONG_SETUP_select {
  Source BP0_LONG_SETUP_logic_signal;
}
ToResetPort BP0_LONG_SETUP_reset_inv {
  ActivePolarity 0;
}
ScanInPort BP0_LONG_SETUP_from_scan_out;
ScanOutPort BP0_LONG_SETUP_to_scan_in {
  Source tdi;
}
ToCaptureEnPort BP0_SHORT_SETUP_capture_en;
ToShiftEnPort BP0_SHORT_SETUP_shift_en;
ToUpdateEnPort BP0_SHORT_SETUP_update_en;
ToSelectPort BP0_SHORT_SETUP_select {
  Source BP0_SHORT_SETUP_logic_signal;
}
ToResetPort BP0_SHORT_SETUP_reset_inv {
  ActivePolarity 0;
}
ScanInPort BP0_SHORT_SETUP_from_scan_out;
ScanOutPort BP0_SHORT_SETUP_to_scan_in {
  Source tdi;
}
ToCaptureEnPort BP1_LONG_SETUP_capture_en;
ToShiftEnPort BP1_LONG_SETUP_shift_en;
ToUpdateEnPort BP1_LONG_SETUP_update_en;
ToSelectPort BP1_LONG_SETUP_select {
  Source BP1_LONG_SETUP_logic_signal;
}
ToResetPort BP1_LONG_SETUP_reset_inv {
  ActivePolarity 0;
}
ScanInPort BP1_LONG_SETUP_from_scan_out;
ScanOutPort BP1_LONG_SETUP_to_scan_in {
  Source tdi;
}
ToCaptureEnPort BP1_SHORT_SETUP_capture_en;
ToShiftEnPort BP1_SHORT_SETUP_shift_en;
ToUpdateEnPort BP1_SHORT_SETUP_update_en;
ToSelectPort BP1_SHORT_SETUP_select {
  Source BP1_SHORT_SETUP_logic_signal;
}
ToResetPort BP1_SHORT_SETUP_reset_inv {
  ActivePolarity 0;
}
ScanInPort BP1_SHORT_SETUP_from_scan_out;
ScanOutPort BP1_SHORT_SETUP_to_scan_in {
  Source tdi;
}
```

```
DataOutPort long_setup {
  Source long_setup_LogicSignal;
}
ScanInterface Client {
  Port tdi;
  Port tdo;
  Port trst;
  Port tms;
  Port tck;
}
ScanInterface BP0_LONG_SETUP {
  Port BP0_LONG_SETUP_capture_en;
  Port BP0_LONG_SETUP_shift_en;
  Port BP0_LONG_SETUP_update_en;
  Port BP0_LONG_SETUP_select;
  Port BP0_LONG_SETUP_from_scan_out;
  Port BP0_LONG_SETUP_to_scan_in;
  Port BP0_LONG_SETUP_reset_inv;
}
ScanInterface BP0_SHORT_SETUP {
  Port BP0_SHORT_SETUP_capture_en;
  Port BP0_SHORT_SETUP_shift_en;
  Port BP0_SHORT_SETUP_update_en;
  Port BP0_SHORT_SETUP_select;
  Port BP0_SHORT_SETUP_from_scan_out;
  Port BP0_SHORT_SETUP_to_scan_in;
  Port BP0_SHORT_SETUP_reset_inv;
}
ScanInterface BP1_LONG_SETUP {
  Port BP1_LONG_SETUP_capture_en;
  Port BP1_LONG_SETUP_shift_en;
  Port BP1_LONG_SETUP_update_en;
  Port BP1_LONG_SETUP_select;
  Port BP1_LONG_SETUP_from_scan_out;
  Port BP1_LONG_SETUP_to_scan_in;
  Port BP1_LONG_SETUP_reset_inv;
}
ScanInterface BP1_SHORT_SETUP {
  Port BP1_SHORT_SETUP_capture_en;
  Port BP1_SHORT_SETUP_shift_en;
  Port BP1_SHORT_SETUP_update_en;
  Port BP1_SHORT_SETUP_select;
  Port BP1_SHORT_SETUP_from_scan_out;
  Port BP1_SHORT_SETUP_to_scan_in;
  Port BP1_SHORT_SETUP_reset_inv;
}
Attribute tessent_instruction_reg = "INSTRUCTION";
Attribute tessent_bypass_reg = "BYPASS";
Attribute tessent_boundary_scan_reg = "BOUNDARY";
Attribute tessent_device_id_reg = "DEVICE_ID";
Attribute tessent_instrument_type = "mentor::jtag_bscan";
Attribute tessent_instrument_subtype = "tap";
Attribute tessent_signature="a693927b83d713ce7521168994205e56";
Enum INSTRUCTION_OPCODES {
  BP2_SHORT_SETUP = 18'bxxxxxx111000xxx010;
  BP3_SHORT_SETUP = 18'bxxxxxx110100xxx010;
  BP0_SHORT_SETUP = 18'bxxxxxx110110xxx010;
  BP1_SHORT_SETUP = 18'bxxxxxx111100xxx010;
```

```
    BP4_SHORT_SETUP=18'bxxxxxx110000xxx010;
    PRELOAD=18'b111111111111111000;
    BYPASS=18'b111111111111111111;
    IDCODE=18'b111111111111111110;
    SAMPLE=18'b111111111111111000;
    CLAMP=18'b111111111111101111;
    EXTEST=18'b111111111111101000;
    BP4_LONG_SETUP=18'bxxxxxx000001xxx010;
    BP2_LONG_SETUP=18'bxxxxxx001001xxx010;
    BP3_LONG_SETUP=18'bxxxxxx000101xxx010;
    BP0_LONG_SETUP=18'bxxxxxx000111xxx010;
    BP1_LONG_SETUP=18'bxxxxxx001101xxx010;
  }
  ScanRegister INSTRUCTION[17:0] {
    ScanInSource tdi;
    CaptureSource 18'bxxxxxxxxxxxxxxxx01;
    ResetValue 18'b111111111111111110;
    RefEnum INSTRUCTION_OPCODES;
  }
  ScanRegister BOUNDARY[41:0] {
    ScanInSource tdi;
  }
  ScanRegister BYPASS {
    ScanInSource tdi;
    CaptureSource 1'b0;
  }
  ScanRegister DEVICE_ID[31:0] {
    ScanInSource tdi;
    CaptureSource 32'b00010001001000110100001001000111;
  }
  ScanMux IRMux SelectedBy fsm.irSel {
    1'b1 : INSTRUCTION[0];
    1'b0 : DRMux;
  }
  ScanMux DRMux SelectedBy INSTRUCTION[17:0] {
    18'b111111111111111000 : BOUNDARY[0];
    18'b111111111111101000 : BOUNDARY[0];
    18'b111111111111001111 : BYPASS;
    18'b111111111111111111 : BYPASS;
    18'b111111111111101111 : BYPASS;
    18'b111111111111111110 : DEVICE_ID[0];
    18'bxxxxxx110110xxx010 : BP0_SHORT_SETUP_from_scan_out;
    18'bxxxxxx111100xxx010 : BP1_SHORT_SETUP_from_scan_out;
    18'bxxxxxx000111xxx010 : BP0_LONG_SETUP_from_scan_out;
    18'bxxxxxx001101xxx010 : BP1_LONG_SETUP_from_scan_out;
  }
  LogicSignal long_setup_LogicSignal {
    (INSTRUCTION == BP0_LONG_SETUP) || (INSTRUCTION == BP1_LONG_SETUP);
  }
  LogicSignal BP0_LONG_SETUP_logic_signal {
    INSTRUCTION == BP0_LONG_SETUP;
  }
  LogicSignal BP0_SHORT_SETUP_logic_signal {
    INSTRUCTION == BP0_SHORT_SETUP;
  }
  LogicSignal BP1_LONG_SETUP_logic_signal {
    INSTRUCTION == BP1_LONG_SETUP;
  }
```

```
    LogicSignal BP1_SHORT_SETUP_logic_signal {
      INSTRUCTION == BP1_SHORT_SETUP;
    }
    Instance fsm Of TOP_bsdl_tap_fsm {
      InputPort tck = tck;
      InputPort tms = tms;
      InputPort trst = trst;
    }
  }
```

Related Topics

read_icl

NameSpace

Syntax

```
NameSpace space_name ;
Module module_name { ... }
```

Description

A NameSpace statement is used to place all subsequent modules in a specific namespace.

This namespace prevents module name collisions, for example, between different vendors, each offering a MemoryBIST module. Each vendor may, for example, use their company name as the namespace, thus avoiding a module name collision.

To reference a particular module, you can either use the UseNameSpace statement or prefix the namespace name before the module name.

Example

```
NameSpace VendorA ;
Module MBIST ;

NameSpace VendorB ;
Module MBIST
```

Then later, you can use, for example:

```
Instance MBIST_inst Of VendorA::MBIST
UseNameSpace VendorB
Instance MBIST_inst Of MBIST
```

Note

Currently Tessent does not support the NameSpace statement.

Related Topics

UseNameSpace

UseNameSpace

Syntax

```
UseNameSpace space_name ;
Module module_name { ... }
```

Description

A UseNameSpace statement is used to reference all subsequent modules from a specific namespace defined earlier through the NameSpace statement.

To reference a particular module, you can either use the UseNameSpace statement or prefix the namespace name before the module name.

Example

```
NameSpace VendorA ;
Module MBIST ;

NameSpace VendorB ;
Module MBIST
```

Then later, you can use, for example:

```
Instance MBIST_inst Of VendorA::MBIST
UseNameSpace VendorB
Instance MBIST_inst Of MBIST
```

Note

Currently Tessent does not support the UseNameSpace statement.

Related Topics

NameSpace

Chapter 7
Command Dictionary

This chapter describes the Tessent Shell commands. For quick reference, the commands appear alphabetically with each beginning on a separate page.

Command Line Syntax Conventions

This manual uses the following command usage line syntax conventions.

Table 7-1. Conventions for Command Line Syntax

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

Command Summary

Table 7-2 contains a summary of the commands described in this manual. For the list of contexts and system modes in which command is available, refer to the command's reference page.

Table 7-2. Command Summary

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

Table 7-2. Command Summary (cont.)

Command	Description
add_config_message	Adds error, warning, or information messages to configuration elements.
add_config_tab	Adds one or many configuration tree tabs in the DFTVisualizer Configuration Data window.
add_control_points	Specifies user-defined control points during test point insertion.
add_core_instances	Adds a core instance to the design by associating/binding the core description currently in memory with the specified core instance(s) in the design.
add_display_data	Adds specified data columns to the Data window of DFTVisualizer, opening the Data window if it is not already open. If data corresponding to an added column is available, it is displayed.
add_display_instances	Displays instances in DFTVisualizer and enables you to trace visually through the design using the Debug or Design window.
add_edt_blocks	Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.
add_edt_configurations	Sets up EDT logic with two compression configurations.
add_edt_connections	Specifies EDT logic connections for the block-level design flow.
add_false_paths	Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.
add_fault_sites	Automatically adds fault site definitions for library cells that don't have UDFM definitions.
add_faults	Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected (actual category is UC).
add_icl_ports	Specifies top design ports that will be added as DataInPort or DataOutPort ports in the ICL file generated during ICL extraction.
add_icl_scan_interfaces	Adds the specified scan interfaces to the new ICL top module that will be created during ICL extraction.

Table 7-2. Command Summary (cont.)

Command	Description
add_iddq_exceptions	Enables you to specify individual sites where the tool does not apply the Iddq restrictions.
add_input_constraints	Constrains primary inputs to specified values.
add_lfsr_connections	Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).
add_lfsr_taps	Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).
add_lfsrs	Adds LFSRs for use as PRPGs or MISRs.
add_lists	Adds pins to the list of pins on which to report.
add_misrs	Specifies the MISR structure and initial seed.
add_nofaults	Places nofault settings either on pin pathnames, pin names of specified instances, or modules.
add_nonscan_instances	Marks specific sequential elements for exclusion from the scan insertion process.
add_nonscan_models	Instructs the tool to ignore all instances of the specified sequential DFT library model when identifying and inserting the required scan elements and test logic into the design.
add_norace_clocks	Defines one or more non-interacting clocks.
add_notest_points	Adds circuit points to the list for exclusion from testability insertion.
add_observe_points	Specifies user-defined observe points during test point insertion.
add_output_masks	Ignores any fault effects that propagate to the primary output pins you name.
add_primary_inputs	Adds a primary input (PI) to the specified nets or pins.
add_primary_outputs	Adds primary outputs to the specified net.
add_processors	Enables the tool to run multiple processes in parallel on multiple machines to reduce runtime.
add_read_controls	Adds an off-state value to specified RAM read control lines.

Table 7-2. Command Summary (cont.)

Command	Description
add_register_value	Creates a static or dynamic variable by allowing you to specify a register identifier, value_name, and, optionally, the state string, value_string.
add_scan_chains	Specifies a name for a pre-existing scan chain within the design.
add_scan_groups	Specifies a group name for a set of pre-existing scan chains in a design. Only one group name can be specified per session.
add_scan_instances	Adds specified instances to the scannable instance list.
add_scan_models	Specifies that the tool is to flag every instance of the named DFT library model for inclusion into the identified scan list.
add_scan_partition	Specifies a grouping of scan cells (a partition) in which scan chains are inserted separately from the remaining scan cells in the design.
add_scan_pins	Declares the name of a scan chain at the top-level module and assigns the corresponding scan input pin, scan output pin, and optionally, scan clock pin you associate with the chain.
add_simdut_fault	Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.
add_simulation_context	Creates a new user-defined simulation context.
add_simulation_force	Forces one or more gate_pin objects to the specified value.
add_sub_chains	Specifies a pre-existing scan chain (subchain) within a module, library model, instance, blackbox, or empty module in a hierarchical design to be stitched into the top-level scan chain during the stitching process.
add_subchain_clocks	Specifies the clock pins for scan chains within modules, library models, instances, blackboxes, or empty modules of a hierarchical design.
add_subchain_group	Adds one subchain group to the system.
add_tied_signals	Adds a value to floating signals or pins.
add_to_collection	Adds objects to a base collection and returns the new collection without affecting the original base collection.
add_write_controls	Specifies the off-state value of the write control lines for RAMs.

Table 7-2. Command Summary (cont.)

Command	Description
analyze_atpg_constraints	Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.
analyze_bus	Analyzes the specified bus gates for contention problems.
analyze_compression	Analyzes the compression for an application in two steps.
analyze_control_signals	Identifies and optionally defines the primary inputs of control signals.
analyze_drc_violation	Generates a schematic in the Debug window of the portion of the design involved with the specified rule violation number.
analyze_fault	Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in DFTVisualizer.
analyze_graybox	Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to “true”. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.
analyze_restrictions	Performs an analysis to automatically determine sources of the problem from a failed ATPG run.
analyze_simulation_mismatches	Analyzes simulation mismatches that occur between test patterns and a simulation performed by a specified external timing simulator.
analyze_test_points	Performs X-bounding analysis.
analyze_wrapper_cells	Identifies shared and dedicated wrapper cells for the primary I/O ports.
analyze_xbounding	Performs X-bounding analysis.
annotate_diagnosis	Adds DFM and RCD data, if available, to existing diagnosis reports.
append_to_collection	Adds one or more objects to a collection referenced by the specified var_name.
apply_specification_defaults	Automatically applies all defaults specified in the DefaultSpecification/DftSpecification wrappers to the DftSpecification wrappers that were created manually.

Table 7-2. Command Summary (cont.)

Command	Description
catch_output	Executes the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.
close_layout	**Layout-Aware Diagnosis Flow:** Closes the Layout Database. Any subsequent diagnosis you perform is non-layout aware. **SVDB Layout Marker Flow:** Releases the Calibre Query Server and closes the layout. If you issue this command, then the tool no longer generates layout markers.
close_pattern_set	Finalizes and closes the currently open pattern_set.
close_tsdb	Makes the content of previously opened TSDB directory invisible to the tool.
close_visualizer	Saves display data and closes DFTVisualizer.
compare_collections	Compares the contents of two collections.
compress_layout	Compresses a Layout Database.
compress_patterns	Compresses patterns in the current test pattern set.
copy_collection	Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.
copy_module	Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.
copy_simulation_context	Copies the simulation values and forces from one simulation context to another.
create_capture_procedures	Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.
create_connections	Connects pin, net, or port objects.
create_dft_specification	Creates the “DftSpecification(design_name,id)” configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.
create_diagnosis_patterns	Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.

Table 7-2. Command Summary (cont.)

Command	Description
create_flat_model	Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the DFTVisualizer debug window.
create_initialization_patterns	Creates RAM initialization patterns and places them in the internal pattern set.
create_instance	Instantiates an instance of a module mod_spec inside a design module in the current design.
create_layout	Generates a diagnosis tool-compatible Layout Database directory from LEF/DEF input files.
create_module	Creates a new design module with no ports.
create_net	Creates a net inside an instance of a design module in the current design.
create_patterns	Performs automatic high-speed test generation and pattern compression.
create_pin	Creates a pin on an instance of a design module.
create_port	Creates a port on the specified design module.
delete_atpg_constraints	Removes state restrictions from the specified pins. You define pin restrictions by using the add_atpg_constraints command.
delete_atpg_functions	Removes the specified function definitions.
delete_bist_capture_ranges	Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.
delete_black_boxes	Undoes the effect of the add_black_boxes command.
delete_browser_data	Removes data from the active tab of the DFTVisualizer Browser window.
delete_capture_handling	Removes the special data capture handling for the specified objects.
delete_capture_procedures	Removes the specified capture procedures from internal memory.
delete_cdp_test	Deletes a test and its associated test patterns from the CDP.
delete_cell_constraints	Removes constraints placed on scan cells.

Table 7-2. Command Summary (cont.)

Command	Description
delete_cell_library	Removes all current cell libraries, the current design including the flattened model, and all tool data that is design-specific.
delete_cell_models	Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.
delete_chain_masks	Removes chain masks from the specified scan chains.
delete_clocks	Removes primary input pins from the clock list.
delete_config_elements	Deletes one or more configuration elements. Only elements which can be added can be deleted.
delete_config_messages	Deletes error, warning or info message that were added to configuration elements using the add_config_message command.
delete_config_tabs	Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using add_config_tab or display_specification.
delete_connections	Disconnects net, pin, or port objects.
delete_core_descriptions	Removes core descriptions that have been read in using the read_core_descriptions command from memory.
delete_core_instances	Removes an instance binding created with the add_core_instances command.
delete_design	Removes the current design including the flattened model, and all tool data that is design-specific.
delete_dfm	Deletes all the DFM rules from the open VDB.
delete_display_data	Removes a data column from the DFTVisualizer Data window.
delete_display_instances	Removes the specified design instances from a DFTVisualizer display window.
delete_edt_blocks	Removes the data for the specified EDT block(s) from the tool's internal database.
delete_edt_configurations	Deletes specified compression configurations from all blocks in a design.

Table 7-2. Command Summary (cont.)

Command	Description
delete_edt_connections	Deletes connections previously specified with the add_edt_connections command.
delete_false_paths	Removes definitions of false paths.
delete_fault_sites	Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.
delete_faults	Removes faults from the current fault list.
delete_flat_model	Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.
delete_icl_modules	Deletes the specified ICL modules from memory.
delete_icl_ports	Undoes the effect of the add_icl_ports command on a specified list of top level ports.
delete_iddq_exceptions	Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.
delete_input_constraints	Removes previously applied constraints from primary input pins.
delete_instances	Deletes the specified instance objects from the current design.
delete_iproc	Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.
delete_layout_verification	Reports or deletes stored layout verification information from the current layout database.
delete_lfsr_connections	Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).
delete_lfsr_taps	Removes the tap positions from a Linear Feedback Shift Register (LFSR).
delete_lfsrs	Removes the specified Linear Feedback Shift Registers (LFSRs).
delete_lists	Removes pins from the pin list the tool monitors and reports on during simulation.
delete_misrs	Deletes MISRs you specify with the add_misrs command.

Table 7-2. Command Summary (cont.)

Command	Description
delete_multicycle_paths	Removes definitions of multicycle paths previously read in using the read_sdc command.
delete_nets	Removes net objects inside an instance of a design module.
delete_nofaults	Removes the nofault settings from either the specified pin or instance/module pathnames.
delete_nonscan_instances	Removes the specified sequential instances from the non-scan instance list.
delete_nonscan_models	Removes from the non-scan model list the specified sequential DFT library models.
delete_norace_clocks	Removes clocks from user-defined non-interacting clock domains.
delete_notest_points	Removes the circuit point which the tool cannot use for testability insertion from the specified pins.
delete_output_masks	Removes the masking of the specified primary output pins.
delete_patterns	Deletes the current pattern set.
delete_pins	Removes pin objects from a design module instance.
delete_ports	Removes port objects from a design module.
delete_primary_inputs	Removes the specified primary inputs from the existing flattened model or, if a flattened model does not yet exist, from the flattened result.
delete_primary_outputs	Removes the specified primary outputs from the existing flattened model or, if a flattened model does not yet exist, from the flattened result.
delete_processors	Removes processor definitions previously specified with the add_processors command.
delete_read_controls	Removes the read control line definitions from the specified primary input pins.
delete_register_value	Deletes all or a specified register value variable specified with the add_register_value command.
delete_scan_chains	Removes the specified scan chain definitions from the scan chain list.
delete_scan_groups	Removes the specified scan chain group definitions from the scan chain group list.

Table 7-2. Command Summary (cont.)

Command	Description
delete_scan_instances	Removes the specified, sequential instances from the user-identified scan instance list.
delete_scan_models	Removes the specified sequential models from the scan model list.
delete_scan_partitions	Deletes the user-specified scan partitions.
delete_scan_pins	Removes any previously assigned scan input, output, and clock names from the specified scan chains.
delete_simdut_fault	Removes the injected stuck-at fault at the specified signal.
delete_simulation_contexts	Deletes one or more user-defined simulation contexts.
delete_simulation_forces	Removes forces from one or more gate_pin objects.
delete_sub_chains	Removes scan subchains previously defined with the add_sub_chains command. Use the report_sub_chains command to display the currently defined scan subchains.
delete_subchain_clocks	Deletes subchain clocks previously defined with the add_subchain_clocks command.
delete_subchain_groups	Removes a scan subchain group.
delete_test_points	Removes the specified test point definitions.
delete_tied_signals	Removes the assigned (tied) value from the specified floating nets or pins.
delete_write_controls	Removes the write control line definitions from the specified primary input pins.
display_specification	Displays the DftSpecification(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.
diagnose_failures	Diagnoses the failures in the specified failure file.
display_specification	Displays the DftSpecification(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.
display_diagnosis_report	Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the DFTVisualizer Debug window.
display_message	Sends a message to the specified message stream.

Table 7-2. Command Summary (cont.)

Command	Description
dofile	Executes the commands contained within the specified file.
echo	Issues a user-defined string to the transcript.
execute_cdp_test	Executes the specified test using the currently connected tester.
execute_gpib_command	Allows you to control and program GPIB equipment.
exit	Terminates the application tool program.
expand_compressed_patterns	Expands compressed 1-hot patterns for better internal failing flop diagnosis.
extract_icl	Checks the ICL connectivity rules between Ijtag instances and extracts the top-level ICL module.
filter_collection	Filters a previously created collection, resulting in a new collection.
find_design_names	Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.
foreach_in_collection	Iterates over each element in a collection.
get_attribute_list	Returns an alphabetically sorted Tcl list of attribute names.
get_attribute_option	Retrieves the current setting of an attribute's option.
get_attribute_value_list	Retrieves the value of an attribute on the specified design objects.
get_cdp_test_list	Returns a Tcl list of tests currently available in the CDP.
get_clock_list	Returns a list of all clocks specified with the add_clocks command.
get_clock_option	Returns the setting of an option specified with the add_clocks command.
get_common_parent_instance	Returns the common ancestor of all instances, pins, or nets found in object_spec.
get_config_elements	Returns a collection of configuration elements or a count of configuration elements when the -count option is used.
get_config_messages	Returns a list of message strings attached to a configuration element.

Table 7-2. Command Summary (cont.)

Command	Description
get_config_selected_nodes	Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the Configuration Data window.
get_config_value	Returns different values associated with a configuration element based on the specified option.
get_context	Introspects the current context.
get_current_design	Returns a collection of one element that specifies the top-level design when previously set.
get_current_mode	Returns the current mode as specified by the set_current_mode command.
get_current_simulation_context	Returns the name of the current simulation context.
get_design_level	Returns the design level previously defined with the set_design_level command.
get_design_sources	Returns the pathnames or file extensions previously specified with the set_design_sources command.
get_dfm_rules	Returns a TCL list of the DFM rules contained in the layout database.
get_dft_cell	Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and desired cell function of a specified set of cells.
get_fanins	Returns a collection of all requested objects found in the fanin of the specified pin, net, or port object.
get_fanouts	Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.
get_fault_type	Returns the current fault model type, as previously specified by the set_fault_type command.
get_gate_pins	Returns a collection of gate_pin objects found below the current design in the flat model.
get_icl_extraction_options	Provides access to the settings specified by the set_icl_extraction_options command.
get_icl_fanins	Returns a collection of all requested objects found in the fanin of the specified pin or port objects.

Table 7-2. Command Summary (cont.)

Command	Description
get_icl_fanouts	Returns a collection of all requested objects found in the fanout of the specified ICL pin, or port objects.
get_icl_instances	Returns a collection of all ICL instances instantiated relative to the current design that match the specified *name_patterns list or -filter expression.
get_icl_modules	Returns a collection of all ICL modules that match the specified name_patterns list or -filter expression.
get_icl_pins	Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified name_patterns list or -filter expression.
get_icl_ports	Returns a collection of all ports on a given module that match the specified name_patterns list or -filter expression.
get_icl_scope	Introspects the current ICL scope.
get_iclock_list	Returns a list of ICL port and pin names on which you have issued an iClock command.
get_iclock_option	Returns the effective or specified source, frequency multiplier, frequency divider, or clock period values for the specified icl_port_or_pin_name.
get_ijtag_instances	Returns the design level previously defined with the set_design_level command.
get_ijtag_instance_option	Returns the value of the selected option for the specified Ijtag instance.
get_ijtag_retargeting_options	Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not issued.
get_insertion_option	Returns the value of any insertion option that is specifiable with the set_insertion_options command.
get_insert_test_logic_options	Retrieves the current prefix that will be used when creating test point logic.
get_instances	Returns a collection of all instances instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the other optional options.

Table 7-2. Command Summary (cont.)

Command	Description
get_iproc_argument_default	Returns the default value for the specified arg_name of the specified proc_name attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_argument_list	Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_body	Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.
get_iproc_list	Returns the current fault model type, as previously specified by the set_fault_type command.
get_logical_library_list	Returns a Tcl list with all logical design library names defined with the set_logical_design_libraries command.
get_license_queue_timeout	Returns the amount of time Tessent Shell will queue for a required license as specified by the previous set_license_queue_timeout command, or its default value of “unlimited’ if the set_license_queue_timeout command was not issued or it was issued with that value.
get_modules	Returns a collection of all modules that match the specified name_patterns list.
get_multiprocessing_option	Returns the value of a single specified variable previously set with the set_multiprocessing_options command.
get_name_list	Returns the name(s) of the specified object(s).
get_nets	Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the other optional options.
get_open_pattern_set	Returns the name of the currently open pattern set.
get_pattern_cycle_count	Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.
get_pattern_set_data	Returns requested details of the internal representation of the specified pattern set.
get_pattern_set_list	Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

Table 7-2. Command Summary (cont.)

Command	Description
get_pattern_set_option	Returns the value of the requested option for the specified pattern_set_name.
get_pins	Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified name_patterns list and subject to filtering imposed by the other optional options.
get_ports	Returns a collection of all hierarchical ports on a given module that match the specified name_patterns list and subjected to filtering imposed by the other options.
get_procfile_name	Returns the absolute path to the scratch directory.
get_resource	Returns true when all resources in the specified resource_list are available.
get_scratch_directory	Returns the absolute path to the scratch directory.
get_simulation_context_list	Returns the available simulation contexts in a Tcl list.
get_simulation_value_list	Returns a Tcl list of the simulation values on the specified gate_pin objects.
get_single_name	Returns a string with the name of the element specified by the object_spec argument.
get_system_mode	Returns the system mode.
get_test_end_icall_list	Returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list containing the iProc and its arguments.
get_test_setup_icall_list	Returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list containing the iProc and its arguments.
get_tool_info	Returns information about the current tool and the current run as a Tcl result.
get_trace_flat_model_option	Introspects the default behavior of the trace_flat_model command.
get_transcript_style	Returns the keyword that specifies the style in which the tool transcripts commands.
get_tsdb_list	Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

Table 7-2. Command Summary (cont.)

Command	Description
get_tsdb_output_directory	Returns the directory used by the process_dft_specification command which is the directory that has been previously set by the set_tsdb_output_directory command.
help	Lists commands and associated usage syntax.
history	Displays a list of previously executed commands.
iApply	Triggers the retargeting of all queued iRead and iWrite commands.
iCall	Calls an iProc registered against the ICL module associated with the specified effective_icl_instance_path.
iClock	Checks whether a controlling clock path from a valid clock source to the specified clock port exists.
iClockOverride	Models how the functional clocking has been programmed.
identify_redundant_faults	Identifies combinationally redundant (RE) faults from among those faults in the current fault list whose redundancy has not been determined yet.
iMerge	Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.
import_dfm	Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).
index_collection	Extracts a single object at a specified index in a collection and creates a new collection containing only that object.
iNote	Inserts a note in the opened pattern set.
insert_test_logic	Inserts the test structures you define into the netlist to increase the design's testability.
intercept_connection	A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.
iPrefix	Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.
iProc	Specifies a PDL procedure that can run later when referenced by the iCall command.

Table 7-2. Command Summary (cont.)

Command	Description
iProcsForModule	Specifies the ICL module for which subsequent iProc commands refer to.
iRead	Adds a read operation to the command queue that will be solved by the next iApply command.
iRelease	Releases the ownership of a resource that has been reserved by iTake.
iReset	Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.
iRunLoop	Creates a vector loop of a given duration.
is_collection	Indicates whether string is a collection handle.
iTake	Used inside of an iProc to reserve a resource (a port, a register or an instance) for exclusive use with this iProc.
iUseProcNameSpace	Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.
iWrite	Adds a write operation to the command queue that will be solved by the next iApply command.
launch_sid_tester	Directs Tessent Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file. Additionally, this command specifies an existing CDP or initializes a new CDP.
lock_current_registration	Locks all previously registered commands and attributes preventing them from being unregistered and modified.
macrotest	Runs the MacroTest utility to automatically generate manufacturing test patterns for embedded logic and memories referred to as macros. This utility is useful for applying specific patterns that test the insides of a macro, thereby improving overall IC test quality.
mark_display_instances	Changes the color of the specified instances in the schematic windows.
migrate_layout	Generates an LDB directory hierarchy that contains the specified preexisting LDB.
move_config_element	Moves a configuration element from one location to another.

Table 7-2. Command Summary (cont.)

Command	Description
move_connections	Turns off transcripting for the specified commands.
no_transcript	Turns off transcripting for the specified commands.
open_layout	Opens an existing Tessent Diagnosis tool-compatible Layout Database directory.
open_pattern_set	Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.
open_tsdb	Makes the content of TSDB directory visible to the tool. The specified TSDB output directory is automatically opened and does not need to be opened with this command.
open_visualizer	Opens the main DFTVisualizer window.
order_patterns	Reorders the internal test pattern set as specified.
printenv	Prints out the values of the Linux variables in the environment.
process_dft_specification	Validates and processes the content contained in a DftSpecification wrapper.
range_collection	Returns one or more adjacent elements from a collection and creates a new collection.
read_cell_library	Loads one or more cell libraries into the tool.
read_config_data	Reads the content of configuration data files or a string into the Tessent Shell environment.
read_core_descriptions	Reads the specified Tessent core description (TCD) files.
read_cpf	Loads a CPF (Common Power Format) file and removes any previously loaded power data. You can use multiple CPF files by using the native CPF “include” command.
read_failures	Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.
read_fault_sites	Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.
read_faults	Updates the current fault list with the faults contained in the specified fault file.
read_flat_model	Loads a flattened model into the tool.

Table 7-2. Command Summary (cont.)

Command	Description
read_icl	Reads ICL files into the internal ICL database.
read_modelfile	Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.
read_patterns	Loads the external pattern set from the specified file.
read_procfile	Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.
read_sdc	Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.
read_sdf	Loads timing data from a Standard Delay Format (IEEE SDF P1497 Draft 0.10) file.
read_upf	Loads a UPF (Universal Power Format) file and removes any previously loaded power data.
read_verilog	Reads one or more Verilog files into the specified or default logical library.
read_vhdl	Loads one or more VHDL files.
read_visualizer_preferences	Reads a DFTVisualizer preferences file and sets current preferences as described in the file.
register_attribute	Registers a new user-defined attribute.
register_tcl_command	Registers a Tcl command and makes it available in the global name space.
remove_from_collection	Removes objects found in obj_spec from a collection, resulting in a new collection.
rename_instance	Renames the leaf name of an instance.
replace_instances	Replaces the module definition of an instance with a new module definition.
report_aborted_faults	Displays information on testable faults that remain undetected (UD) after the ATPG process.
report_atpg_constraints	Displays all the current ATPG state restrictions and the pins on which they reside.
report_atpg_functions	Displays all the current ATPG function definitions.

Table 7-2. Command Summary (cont.)

Command	Description
report_atpg_timing	Reports the current ATPG timing setup.
report_attributes	Creates a human readable report for the set of attributes based on the usage options.
report_bist_capture_ranges	Displays a list of the currently defined BIST capture ranges.
report_black_boxes	Displays information on blackboxes and undefined models.
report_bus_data	Displays the bus data information for either an individual bus gate or for the buses of a specific type.
report_bypass_chains	Displays the defined EDT bypass scan chain connections.
report_capture_handling	Displays any special data capture handling currently in use.
report_capture_procedures	Displays the cyclized information for a specified named capture procedure.
report_cell_constraints	Displays a list of the constrained cells.
report_cell_models	Displays a list of either all cell models or the DFT library models associated with the specified cell type.
report_chain_masks	Reports the list of masked scan chains and their associated unload value.
report_clock_controller_pins	Reports the clock controller pins specified with the set_clock_controller_pins command.
report_clock_controls	Displays current configuration information for clock control definitions.
report_clock_domains	Displays all clocks and their corresponding clock domain numbers.
report_clock_gating	Reports the clock gating instances that were identified as having unconnected ports and were connected to either the scan enable signal or to a user-specified signal or, reports the unconnected ports of the specified clock gating cells only.
report_clocks	Displays a list of all user-defined clocks and SDC-defined clocks.
report_compactor_connections	Displays defined scan chain connections for each EDT scan channel output.
report_context	Reports the current context as specified by the set_context command and any inferred subcontexts.

Table 7-2. Command Summary (cont.)

Command	Description
report_control_signals	Displays the rules checking results for control signals consisting of clocks added with the add_clocks command and pins identified for gating scannable memory elements with test logic.
report_core_instances	Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.
report_design_sources	Reports the pathnames or file extensions previously specified with the set_design_sources command.
report_dfm_rules	Returns a tabular report of the DFM rules contained in the layout database.
report_dft_check	Generates scannability check information for all non-scan instances in the design, or within a specific hierarchical instance.
report_diagnosis	Writes the scan diagnosis report to standard I/O.
report_display_instances	Shows netlist information for the specified instances displayed in the Debug, Design, or Data window.
report_drc_rules	Displays either a summary of DRC violations (fails) or violation occurrence message(s).
report_edt_blocks	Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.
report_edt_configurations	Displays configuration information for the current compression logic.
report_edt_connections	Reports pin connections made during the top-level integration session.
report_edt_finder	Reports information about the EDT logic contained in a design.
report_edt_instances	Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.
report_edt_lockup_cells	Displays a tabular report of the lockup cells in the EDT logic.
report_edt_pins	Displays the currently defined names and inversion status of EDT channel and control pins.

Table 7-2. Command Summary (cont.)

Command	Description
report_environment	Displays the current value(s) of many of the most common “set_...” commands.
report_external_simulator	Displays the shell command the tool will use to invoke an external simulator for verifying patterns.
report_failures	Simulates the current design and test patterns and reports the failures in a pattern-based format.
report_false_paths	Displays false paths previously defined with the add_false_paths command.
report_fault_sites	Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.
report_faults	Displays fault information from the current fault list.
report_feedback_paths	Displays a report of currently identified feedback paths.
report_flattener_rules	Displays either a summary of all the flattening rule violations or the data for a specific violation.
report_gates	Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.
report_icl_extraction_options	Provides a human readable report of the ICL extraction options.
report_icl_modules	Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.
report_graybox_statistics	Reports the statistics gathered by graybox analysis.
report_id_stamp	Displays the unique identifier that the tool assigns each internal pattern set.
report_iddq_exceptions	Displays current exceptions to Iddq restrictions.
report_ijtag_instances	Reports options of instances for which there is a matched ICL module.
report_ijtag_logical_connections	Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. If no pin or port connections are specified, all logical connections are listed.

Table 7-2. Command Summary (cont.)

Command	Description
report_ijtag_retargeting_options	Reports the type and value of each option of the previous set_ijtag_retargeting_options command or their default values if the set_ijtag_retargeting_options command was not issued.
report_input_constraints	Displays the present constraint status of primary input pins.
report_layout_rules	Provides detailed reporting of layout rule violations.
report_lbist_configuration	Reports the global logicBIST controller configuration parameters.
report_lbist_pins	Reports the LogicBIST controller pins using the set_lbist_pins command.
report_lfsr_connections	Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.
report_lfsrs	Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).
report_licenses	Reports the license features currently checked out by the tool.
report_lists	Displays the list of pins whose values the tool will report during simulation.
report_loops	Displays information about circuit loops.
report_lpct_configuration	Reports the current low pin count test controller (LPCT) parameters to the standard display output.
report_lpct_pins	Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.
report_measure_cycles	Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.
report_mismatch_sources	Displays the sources of simulation mismatches.
report_misrs	Reports the MISRs you have added to your design.
report_misr_connections	Reports MISR connections.
report_module_matching	Reports design modules and their matching ICL modules if the set_current_design command has been executed in ICL extraction mode prior.

Table 7-2. Command Summary (cont.)

Command	Description
report_module_matching_options	Reports the current settings defined by the set_module_matching_options command.
report_multicycle_paths	Displays multicycle path definitions you previously read into the tool using the read_sdc command.
report_multiprocessing_options	Displays the values of all variables used when executing multiprocessing commands for ATPG or fault simulation.
report_nofaults	Displays the nofault settings for the specified pin pathnames or pin names of instances.
report_nonscan_cells	Displays the non-scan cells whose model type you specify.
report_nonscan_models	Displays the sequential non-scan model list.
report_norace_clocks	Displays a list of all user-defined non-interacting clocks and their clock domain numbers.
report_notest_points	Displays all the circuit points for which you do not want the tool to insert controllability and observability.
report_output_masks	Displays a list of the currently masked primary output pins.
report_pattern_filtering	Displays the current pattern filtering configuration.
report_pattern_sets	Creates a report of a specified pattern set or of all pattern sets.
report_patterns	Displays information about patterns.
report_power_data	Reports the power data currently loaded in the system from either a read_cpf or read_upf command.
report_power_metrics	Displays shift and capture power metrics for specified test patterns.
report_primary_inputs	Displays a list of the primary inputs of a circuit.
report_primary_outputs	Displays the specified primary outputs.
report_procedures	Displays the specified procedure.
report_processors	Displays information about the processors used for the current multiprocessing environment.
report_read_controls	Displays all of the currently defined read control lines.
report_register_value	Reports register information for each register name.

Table 7-2. Command Summary (cont.)

Command	Description
report_resources	Displays the machine resources used by the active tool session.
report_scan_cells	Displays a report on the scan cells that reside in the specified scan chains.
report_scan_chains	Displays a report on all the current scan chains.
report_scan_enable	Reports on scan_enable signals and associated scan chains.
report_scan_groups	Displays a report on all the current scan chain groups.
report_scan_models	Displays the sequential scan models currently in the scan model list.
report_scan_partitions	Displays the scan partitions.
report_scan_pins	Displays all previously assigned scan input, output, and clock names.
report_scan_volume	Displays the volume of scan data used by the pattern set.
report_seq_transparent_procedures	Displays a list of seq_transparent test procedures along with the associated data that you specify.
report_sequential_fault_depth	Displays estimates of maximum test coverage possible with different sequential depth settings.
report_sequential_instances	Reports information and testability data for the sequential instances in the design.
report_shift_registers	Reports the identified shift registers in the design after switching to analysis mode.
report_simulation_contexts	Lists the available simulation contexts and indicates the current simulation context.
report_simulation_forces	Lists the active forces on the specified gate_pin objects.
report_statistics	Displays a detailed statistics report to the screen.
report_sub_chains	Displays a report on the scan subchains.
report_subchain_clocks	Reports on subchain clocks defined with the add_subchain_clocks command.
report_subchain_groups	Displays a report on the subchain groups.
report_test_logic	Displays the test logic that the tool added during the scan insertion process.

Table 7-2. Command Summary (cont.)

Command	Description
report_test_points	Displays both user-defined and system-defined test points.
report_test_stimulus	Displays the stimulus necessary to satisfy the specified set, write, or read conditions.
report_testability_data	Analyzes collapsed faults for the specified fault class and displays the analysis.
report_tied_signals	Displays a list of the tied floating signals and pins.
report_timeplate	Displays the specified timeplate.
report_udfm_statistics	Reports detailed fault coverage based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU faults.
report_version_data	Displays the current software version information.
report_wrapper_cells	Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.
report_write_controls	Displays the currently defined write control lines and their off-states.
reset_attribute_value	Resets an attribute to its default value for the design objects specified in obj_spec.
reset_au_faults	Reclassifies the faults in certain untestable categories.
reset_bypass_chains	Removes user-specified EDT bypass chain connections.
reset_compactor_connections	Removes user-specified EDT compactor connections.
reset_design	Clears all design setup data but keeps the design and library in memory.
reset_di_faults	Reclassifies DI faults to the UC category.
reset_open_pattern_set	Clears the content of the currently open pattern set.
reset_state	Resets the circuit status.
restore_design	Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.
return_output	Accepts one or more tool command lines and returns the command output as a Tcl result.
select_display_instances	Selects the specified objects in the Debug and Design window(s).

Table 7-2. Command Summary (cont.)

Command	Description
set_abort_limit	Specifies the abort limit for the test pattern generator.
set_atpg_compression	Specifies for the ATPG to perform dynamic pattern compression.
set_atpg_fill	Specifies how the tool should fill unspecified bits in ATPG patterns.
set_atpg_limits	Specifies the ATPG process limits at which the tool terminates the ATPG process.
set_atpg_timing	Enables and defines parameters for timing-aware ATPG and fault simulation.
set_attribute_options	Modifies options of any registered user-defined attribute.
set_attribute_value	Sets an attribute's value for the objects specified in object_spec.
set_au_analysis	Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.
set_bidi_gating	Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.
set_bist_chain_test	Disables all capture clock activity for the specified number of patterns in the fault simulation during execution of the chain test.
set_bist_debug	Sets up a trace of LFSR(s) value during a pattern's shift cycles.
set_bist_debug	Sets up a trace of LFSR(s) value during a pattern's shift cycles.
set_bist_trace	Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.
set_bus_handling	Specifies the bus contention results that you desire for the identified buses.
set_bus_simulation	Specifies whether the tool uses global or local bus simulation analysis.
set_bypass_chains	Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.

Table 7-2. Command Summary (cont.)

Command	Description
set_capture_clock	Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.
set_capture_handling	Specifies how the tool globally handles the data capture of state elements that have C3 and C4 rule violations.
set_capture_procedures	Enables or disables the tool's use of a specified subset of named capture procedures when generating test patterns.
set_cell_library_options	Sets the specified options for library parsing and design elaboration.
set_cell_model_mapping	Overrides the non-scan to scan model mapping defined by the tool.
set_chain_test	Specifies binary sequences for chain test.
set_checkpointing_options	Enables checkpointing and specifies the associated options.
set_clock_controller_pins	Specifies the connection information for the clock controller pins.
set_clock_controls	Enables and disables the clock control definitions in the test procedure file.
set_clock_gating	Specifies clock gating cells whose unconnected scan enable ports need to be connected to either the scan enable signal or a specified signal (pin).
set_clock_gating_enable	Associates clocks that control clock gating devices with enable signal drivers.
set_clock_off_simulation	Enables or disables the simulation where all clock primary inputs are at their "off" value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.
set_clock_restriction	Specifies whether ATPG can create patterns with more than one active capture clock.
set_compactor_connections	Specifies the scan chains for the tool to connect to a particular scan channel output.
set_contention_check	Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.
set_context	Specifies the current usage context of Tessent Shell.

Table 7-2. Command Summary (cont.)

Command	Description
set_current_design	Specifies the top level of the design for all subsequent commands until reset by another execution of this command.
set_current_edt_block	Directs the tool to apply certain subsequent commands to a particular EDT block only.
set_current_edt_configuration	Sets a particular compression configuration for the tool to use within the current design block.
set_current_mode	Specifies the current mode.
set_current_simulation_context	Sets the current simulation context.
set_decision_order	Specifies how the ATPG selects gate input or output nodes for evaluation during ATPG.
set_defaults_value	Sets a property inside one of the DefaultsSpecification/DftSpecification (policy) wrappers.
set_design_include_directories	Specifies one or more directory paths that contain `include files for use by subsequent read_verilog commands.
set_design_level	Specifies the current usage context of Tessent Shell.
set_design_macros	Specifies one or more defined macros for use by subsequent commands.
set_design_sources	Specifies where the tool should look for the definition of undefined modules in the list of files specified by the read_verilog and read_icl commands.
set_dft_enable_options	Enables or disables the control or observe points.
set_diagnosis_options	Specifies the type of diagnosis performed by the diagnose_failures command.
set_display	Sets the DISPLAY environment variable from the tool's command line.
set_dofile_abort	Specifies whether the tool aborts or continues dofile execution if it detects an error condition.
set_drc_handling	Specifies how the tool handles design rule violations.
set_driver_restriction	Specifies whether the tool allows multiple drivers on buses and multiple active ports on gates.
set_edt_finder	Enables the tool to automatically identify EDT logic and update scan chain pins during test pattern generation.

Table 7-2. Command Summary (cont.)

Command	Description
set_edt_instances	Specifies the location in which the tool places the EDT logic.
set_edt_mapping	Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.
set_edt_options	Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.
set_edt_pins	By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.
set_edt_power_controller	Generates and inserts a power controller in the compression logic to enable low-power shift.
set_external_capture_options	Specifies the number of tester cycles to be used in the capture window of all patterns.
set_external_simulator	Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.
set_fails_report	Specifies whether the design rules checker displays clock rule failures.
set_fault_mode	Specifies whether the fault mode is collapsed or uncollapsed.
set_fault_protection	Specifies for the tool to protect detected faults from manipulation by subsequent tool commands.
set_fault_sampling	Specifies the fault sampling percentage used for circuit evaluation or scan identification.
set_fault_subclass_analysis	Turns on or off the analysis for the specified fault class(es).
set_fault_type	Specifies the fault model type for ATPG.
set_file_compression	Controls whether the tools read and write files with .Z or .gz extensions as compressed files (the default).
set_flattener_rule_handling	Specifies how the tool globally handles flattening violations.

Table 7-2. Command Summary (cont.)

Command	Description
set_gate_level	Specifies the hierarchical level of gate reporting and displaying.
set_gate_report	Specifies the information displayed by the report_gates command.
set_gzip_options	Specifies GNU gzip options to use with the GNU gzip command.
set_icl_extraction_options	Customizes the behavior of ICL extraction.
set_icl_scan_interface_ports	Sets the names of the ports in the specified scan interface previously added by add_icl_scan_interfaces that ICL extraction will create in the new ICL top module.
set_iddq_checks	Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.
set_iddq_strobe	Specifies on which patterns (cycles) the tool will simulate IDDQ measurements.
set_ijtag_instance_options	Sets options on instances for which there is a matched ICL module.
set_ijtag_retargeting_options	Enables you to configure different aspects of ijtag retargeting.
set_insertion_options	Specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.
set_insert_test_logic_options	Specifies a prefix that overrides the default prefix “ts_” that is used when creating the test point logic.
set_internal_fault	Specifies whether the tool allows faults within or only on the boundary of library models.
set_internal_name	Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.
set_io_mask	Modifies the behavior of bidirectional pins so that their expected values will always be X during test cycles in which the primary input portion of the bidirectional pin is being forced.
set_latch_handling	Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.
set_lbist_controller_options	Specifies global options to configure the logicBIST controller.

Table 7-2. Command Summary (cont.)

Command	Description
set_lbist_instances	Specifies the instance in which the logicBIST controller or single chain mode logic is placed.
set_lbist_pins	Specifies the connection information for logicBIST controller pins.
set_lbist_power_controller_options	Specifies creation of the low-power shift controller for LogicBIST.
set_learn_report	Specifies whether the report_gates command can display the learned behavior for a specific gate.
set_lfsr_seed	Sets the initial value of a PRPG or a MISR.
set_lfsrs	Changes the shift_type and tap_type default setting for the add_lfsrs and add_lfsr_taps commands.
set_license_queue_timeout	Specifies the amount of time Tessent Shell will queue for a required license if it is not immediately available.
set_list_file	Specifies the name of the list file into which the tool places the pins' logic values during simulation.
set_lockup_cell	Enables the automatic insertion of lockup cells.
set_logfile_handling	Specifies for the tool to direct the transcript information to a file.
set_logical_design_libraries	Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.
set_loop_handling	Specifies how the tool handles feedback networks.
set_lpct_condition_bits	Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.
set_lpct_controller	Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.
set_lpct_instances	Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.
set_lpct_pins	Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.
set_macrotest_options	Prepares a black box macro to allow MacroTest conversion of patterns at the boundaries of that instance.

Table 7-2. Command Summary (cont.)

Command	Description
set_misr_connections	Describes the connections between the scan chain outputs and the MISR.
set_module_matching_options	Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL module to a design module during ICL extraction.
set_multiple_detection	Enables the multiple detection of faults.
set_multiprocessing_options	Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.
set_name_prefix	Explicitly defines the default names for nets and instances, and reports current or modified settings.
set_net_dominance	Specifies the fault effect of bus contention on tri-state nets.
set_net_resolution	Specifies the behavior of multi-driver nets.
set_number_shifts	Sets the number of shifts for loading or unloading the scan chains.
set_observation_point	Specifies the observation point for random pattern fault simulation.
set_output_masks	Ignores any fault effects that propagate to the primary output pins you select.
set_parallel_load_subchains	Specifies whether the internal parallel access functionality is enabled or disabled.
set_pathdelay_holdpi	Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.
set_pattern_buffer	Stores run-time pattern data in temporary files in the directory specified.
set_pattern_classification	Specifies the order in which to store test patterns.
set_pattern_filtering	Creates a temporary set of sampled scan patterns.
set_pattern_source	Specifies the source of the test patterns.
set_pattern_type	Specifies the type of pattern(s) used by ATPG simulation.
set_physical_translation	Translates special Verilog characters to SPICE-readable characters Calibre applications can read.
set_possible_credit	Specifies the percentage of credit that the tool assigns possible-detected faults.

Table 7-2. Command Summary (cont.)

Command	Description
set_power_control	Enables low-power ATPG and sets up a switching and/or capture threshold.
set_power_domain	Specifies the number or the length of scan chains to be generated for the given power domains.
set_power_metrics	Calculates shift and capture power during test pattern generation and simulation.
set_procedure_cycle_checking	Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.
set_procedure_retargeting_options	Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.
set_procfile_name	Specifies a new procedure file for the tool to process at a later time.
set_ram_initialization	Specifies whether to initialize RAM and ROM gates that do not have initialization files.
set_ram_test	Specifies the mode for RAM testing with random patterns.
set_random_atpg	Specifies whether the tool uses random patterns during ATPG.
set_random_clocks	Specifies whether Tessent FastScan uses combinational or clock_sequential patterns for random pattern simulation.
set_random_patterns	Specifies the number of random patterns that the tool simulates.
set_register_value	Provides a method for specifying the register value variable outside of Setup mode.
set_registered_io	Turns on the automatic registration of the primary I/O pins of the design.
set_relevant_coverage	Determines which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.
set_scan_enable	Assigns scan_enable signals to specific scan chains.
set_scan_enable_sharing	Divides all scan chains into specified groups and assigns a unique scan_ enable signal to each group.

Table 7-2. Command Summary (cont.)

Command	Description
set_scan_insertion	Specifies the input control signals' drivers used by the insert_test_logic command.
set_scan_pins	Changes the scan-in or scan-out pin naming parameters to index or bus format.
set_screen_display	Specifies whether the tool writes the transcript to the session window.
set_shadow_check	Specifies whether the tool will identify sequential elements as a "shadow" element during scan chain tracing.
set_shift_register_identification	Enables/disables shift register identification. The default is ON.
set_simulation_options	Controls parallel simulation behavior.
set_skewed_load	Specifies whether Tessent FastScan includes a skewed load in the patterns.
set_split_capture_cycle	Controls whether the tool updates simulation data between clock edges.
set_stability_check	Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.
set_static_learning	Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.
set_system_mode	Specifies the operational state you want the tool to enter.
set_test_end_icall	Adds an iCall to the start of the test_end procedure.
set_test_logic	Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.
set_test_point_analysis_options	Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.
set_test_point_insertion_options	Sets the following test point insertion related parameters: - name of the test point enable signals. - number of test points that will be observed using a single flip-flop. - connect the clock port of all newly inserted flip-flops to a global test_clock signal instead of re-using existing clock signals.
set_test_setup_icall	Adds an iCall to the end of the test_setup procedure.

Table 7-2. Command Summary (cont.)

Command	Description
set_tied_signals	Changes the default value for floating pins and floating nets which do not have assigned values.
set_timing_exceptions_handling	Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command.
set_tla_loop_handling	Specifies how to simulate feedback loops that contain transparent latches (TLAs).
set_trace_report	Specifies whether the tool displays gates in the scan chain trace.
set_transcript_style	Specifies the style in which the tool transcripts commands.
set_transient_detection	Specifies whether the tool detects all zero width events on the clock lines of state elements.
set_transition_holdpi	Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.
set_tristate_gating	Specifies how tri-state devices are controlled during scan chain shifting.
set_tsdb_output_directory	Sets the tsdb output directory used by the process_dft_specification command.
set_visualizer_logging	Writes the commands entered into the Transcript window to the file specified by the enhanced_dofile argument.
set_visualizer_preferences	Controls a subset of DFTVisualizer preferences for the Debug and Design windows.
set_wrapper_chains	Specifies wrapper chains for scan insertion.
set_xbounding_options	Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.
set_xclock_handling	Specifies whether the sequential element model outputs X when any of its clock inputs become X.
set_z_handling	Specifies the simulation handling for high impedance signals on internal and external tri-state nets.
set_zhold_behavior	Specifies whether ZHOLD gates retain their state values.
setenv	Sets a shell environment variable within the tool environment.

Table 7-2. Command Summary (cont.)

Command	Description
shutdown_sid_tester	Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.
simulate_clock_pulses	Pulses one or more clocks within the current simulation context.
simulate_forces	Simulates the queued forces in the current simulation context.
simulate_patterns	Performs simulation by applying the specified pattern source.
sizeof_collection	Returns the number of objects in a collection.
sort_collection	Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.
system	Passes the specified command to the operating system for execution.
trace_flat_model	Traces the flat model taking into account the current simulation context.
uncompress_layout	Uncompresses a Layout Database.
uniquify_instances	Uniquifies the module definition of the specified instance objects found in obj_spec.
unmark_display_instances	Returns the specified marked instances to their original color. Color marking occurs with the mark_display_instances command.
unregister_attribute	Unregisters a user-defined attribute that was registered using the register_attribute command.
unregister_tcl_command	Unregisters the command called command_name that was previously registered using the register_tcl_command.
unselect_display_instances	Unselects the specified objects in the Debug and Design window(s).
unsetenv	Unsets a shell environment variable within the tool environment.
update_implication_detections	Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

Table 7-2. Command Summary (cont.)

Command	Description
verify_patterns	Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.
write_atpg_setup	Writes a test procedure file and the dofile file for the existing scan chains.
write_core_description	Writes out the core description corresponding to the current chip level.
write_core_timing_constraints	Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.
write_design	Writes the current design in Verilog netlist format to the specified file.
write_diagnosis	Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.
write_edt_files	Creates the files that implement the EDT logic.
write_failing_paths	Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.
write_failures	Injects faults and/or writes failing pattern results to a file.
write_fault_sites	Writes path definitions or bridge entries from the internal list to a file.
write_faults	Writes fault information from the current fault list to a file.
write_flat_model	Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file.
write_icl	Writes out ICL modules created and/or read in with the read_icl command to the specified file.
write_loops	Writes a list of all the current loops to a file.
write_modelfile	Writes all internal states for a RAM or ROM gate into the file that you specify.
write_patterns	Saves the current test pattern set to a file in a specified format.
write_primary_inputs	Writes primary inputs to the specified file.
write_primary_outputs	Writes primary outputs to the specified file.

Table 7-2. Command Summary (cont.)

Command	Description
write_procedure_testbench	Writes a test bench for verifying correct operation of internal signals used in procedures.
write_procfile	Writes existing procedure and timing data to the named test procedure file.
write_scan_order	Writes out the scan chains in the design using the DEF language.
write_scan_setup	Writes out a dofile containing all necessary steps for scan insertion.
write_subchain_setup	Writes the appropriate add_sub_chains commands to a file so that the tool can understand the pre-existing scan subchains at the top-level of this module.
write_test_point_dofile	Writes out a dofile containing the test points.
write_visualizer_dofile	Writes a dofile containing the commands needed to recreate the current instance and data sets displayed in the DFTVisualizer windows.
write_visualizer_preferences	Writes the current DFTVisualizer preference settings to the specified file.
write_window_contents	Saves the current content of the specified DFTVisualizer window to the specified file.

Command Descriptions

This section describes, in alphabetical order, each command available in Tessent Shell. The beginning of each command description shows the contexts and modes that support the command and provides the following information:

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

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

add_ambiguous_paths

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

Mode: analysis

Usage

add_ambiguous_paths {***path_name*** | **-All**} [-Max_paths *number*]

Description

Specifies to derive multiple paths from ambiguous paths when there is path or edge ambiguity.

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

For more information on path delay faults, see "Creating a Path Delay Test Set" in the *Tessent Tessent Scan and ATPG User's Manual*.

Arguments

- ***path_name***

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

- **-All**

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

- -Max_paths *number*

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

Note

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

Example

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

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

Related Commands

delete_fault_sites
read_fault_sites
write_patterns

add_atpg_constraints

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

Mode: setup, analysis

Usage

add_atpg_constraints {**0** | **1** | **Z**} {***pin_pathname*** | ***gate_id#*** | {***function_name*** |
{**-Cell** ***cell_name*** {***pin_name***…}}}… [-Instance {*object_expression*…}]
[-MODUle {*module_name* …}]}… [-DYnamic | -STatic]
[-NOCapclock_check | -Load_only]

Description

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

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

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

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

Note

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

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

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

Arguments

- **0 | 1 | Z**

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

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

- ***pin_pathname***

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

- ***gate_id#***

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

- ***function_name***

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

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

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

- -Instance *object_expression*

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

- -MODUle *module_name*

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

- -DYnamic

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

- -STatic

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

- -NOCapclock_check

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

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

- -Load_only

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

Example 1

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

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

Example 2

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

```
add_atpg_functions mux_1hotse1 SELECT1 -cell imux3sux2 S2 S1 S0
add_atpg_constraints 1 mux_1hotsel -instance core1
```

Example 3

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

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

Example 4

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

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

Related Commands

add_atpg_functions
delete_atpg_constraints
report_atpg_constraints

add_atpg_functions

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

Mode: setup, analysis

Usage

add_atpg_functions ***function_name type*** {***pin_pathname*** | ***gate_id#*** | ***existing_function_name*** | {**-Cell** ***cell_name*** {***pin_name*** | **-Function** ***type pin_name***...}...}}...

Description

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

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

Arguments

- ***function_name***

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

- ***type***

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

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

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

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

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

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

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

- ***pin_pathname***

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

- ***gate_ID#***

  A repeatable integer that specifies the gate identification number.

- ***existing_function_name***

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

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

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

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

  For information on how this switch is used, see Example 2.

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

Example 1

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

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

Example 2

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

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

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

where A, B, C, and D are inputs of the OR4 cell. Normally, you can define two functions, f1 and f2, then add a third function, f3, that refers to f1 and f2. However, when you use the -Cell switch with the add_atpg_functions command, this is not possible. You can use the -Function switch to define up to two levels of functions using the -Cell switch. The syntax is as follows:

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

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

Example 3

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

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

Example 4

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

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

Related Commands

add_atpg_constraints
delete_atpg_functions
report_atpg_functions

add_bist_capture_range

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

add_bist_capture_range ***atpg_capture_proc_name*** [***start_value stop_value***]

Description

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

Hybrid TK/LBIST Flow Usage

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

Additionally, the command is automatically added to the LogicBIST fault simulation dofile written out during IP generation. For usage information, refer to "LogicBIST Fault Simulation and Pattern Creation" in the *Hybrid TK/LBIST Flow User's Manual*.

Arguments

- ***atpg_capture_proc_name***

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

- ***start_value***

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

- ***stop_value***

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

Example

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

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

Related Commands

delete_bist_capture_ranges

report_bist_capture_ranges

set_lbist_controller_options

add_black_boxes

Context: all contexts

Mode: setup

Usage

add_black_boxes {{{-**Instances** ***ins_pathnames*** | -**Modules** ***module_names***}
[0 | 1 | X | Z]} [-Pin *pinname* {0 | 1 | X | Z}]…} | {-**Auto** [0 | 1 | X | Z]}
[-FAUlt_boundary | -NOFAUlt_boundary | -NO_Boundary]

Description

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

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

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

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

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

Tip: For optimal performance, issue all add_black_boxes commands before issuing any other commands.

Arguments

- -**Instances** ***ins_pathnames***

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

Note

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

- **-Modules *module_names***

  A switch and string pair that specifies to blackbox every instance of the Verilog module or ATPG library model specified by the module_names argument. The module_names argument can be an instance, a Tcl list of instances, or a collection of instances.

- **-Auto**

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

Note

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

- 0 | 1 | X | Z

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

- -Pin *pinname* 0 | 1 | X | Z

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

- -FAUlt_boundary

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

- -NOFAUlt_boundary

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

- -NO_Boundary

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

Note

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

Example 1

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

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

Example 2

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

```
//  Compiling library  ...
//  Reading Verilog Netlist ...
//  Reading Verilog file dfta_out/pipe_scan.v
//  Finished reading file dfta_out/pipe_scan.v
//  WARNING: Following modules are undefined:
//      ao21
//      and02
//  Use "add_black_boxes -auto" to treat these as black boxes.
```

add_black_boxes -auto

Example 3

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

add_black_boxes -modules macro1 -fault_boundary

add_black_boxes -modules macro1 -nofault_boundary

Example 4

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

add_black_boxes -modules [get_modules -filter mark_black_box]

Example 5

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

```
add_black_boxes -instances [get_instances -filter mark_black_box]
```

Example 6

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

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

Related Commands

add_tied_signals
delete_black_boxes
report_black_boxes

add_browser_data

Context: all contexts

Mode: setup, analysis

Usage

add_browser_data {{**Gates** | **Primitives**} | {**Drc** [-All_fails | *rule_id*]} |
{**FAults** [-ALl | -Total | -Undet | *fault_class*]} | **T_coverage** | **F_coverage** | **Loss_TC** | **RELevant_test_coverage** | **ATpg_effectiveness}**
[-Tab Hierarchy | Library | Clocks]

Description

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

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

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

Use the delete_browser_data command to delete data columns.

Arguments

- **Gates**

  Required literal that displays the number of Verilog primitives or ATPG library models in each instance listed in the Hierarchy tab of the Browser window.

- **Primitives**

  Required literal that displays the number of ATPG library primitives in each instance listed in the Hierarchy tab of the Browser window.

- **Drc**

  Required literal that displays DRC violations for instances listed in the Hierarchy tab of the Browser window. By default all DRC violations display. Options include:

  rule_id
  String that specifies a design rule to display statistics for. The *rule_id* must match the exact violation identification (ID) of the design rule. For a list of rule violation IDs, refer to "Design Rule Checking."

- **FAults (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

-Total
Switch that lists the number of total faults for all classes. This switch is valid in the Hierarchy and Clocks tabs of the Browser window.

-Undet
Switch that specifies to list the faults of the undetected (UD) class. This switch is valid in the Hierarchy tab of the Browser window.

fault_class
String that specifies a fault class to display faults for. This switch is valid in the Hierarchy and Clocks tabs of the Browser window. For a summary of available fault classes, refer to the report_faults command.

-ALl
Switch that specifies to display all faults classes. This option displays the following columns in the Browser window: UO, DS, DI, PT, UU, and AU. This switch is valid in the Hierarchy and Clocks tabs of the Browser window. This is the default.

- **F_coverage (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

- **T_coverage (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

- **Loss_TC (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

- **RELevant_test_coverage (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

- **ATpg_effectiveness (Tessent FastScan, Tessent TestKompress, and Tessent Diagnosis only)**

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

- -Tab Hierarchy | Library | Clocks

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

Examples

The following example adds a column displaying the number of primitives in each instance and as many additional columns as there are design rules that were violated during DRC, with each of the latter showing the number of violations of each particular rule. Because the -Tab argument is not specified, by default, the column is added to the active tab.

```
add_browser_data primitives drc
```

The next example clears the active tab of the data columns added in the previous example, and adds two columns that list the total number of C2 and C3 DRC violations:

```
delete_browser_data primitives drc
add_browser_data drc c2 drc c3
```

Related Commands

delete_browser_data

add_capture_handling

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

Mode: setup, analysis

Usage

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

Description

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

Specifies the data capturing behavior for the given state element.

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

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

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

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

Arguments

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

- **Old**

  A literal specifying that the source state elements determine their output values for data capture by using the data that existed prior to the current clock cycle. The tool then passes

the data on to the source state element's sink state elements. This option is the default behavior upon invocation of Tessent FastScan and Tessent TestKompress.

- **New**

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

- **X**

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

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

- ***gate_id#***

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

- ***pin_pathname***

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

- ***instance_name***

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

Note

The instance name parameter is only valid for RAM's and FF's.

- **-Cell *cell_name***

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

- -SInk

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

- -SOurce

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

Example

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

add_capture_handling new 1158 -source
set_capture_handling -te new

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

Related Commands

delete_capture_handling
report_capture_handling
set_capture_handling

add_cdp_test

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

Mode: setup, analysis

Usage

add_cdp_test ***test_name*** **-pattern** ***pattern_file***

Description

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

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

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

Arguments

- ***test_name***

  A required string that specifies the name of the test.

- **-pattern** ***pattern_file***

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

Example

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

```
add_cdp_test test1 -pattern ./patterns/pat1.svf
```

Refer to “Executing an IJTAG Test Pattern” in *Tessent SiliconInsight User’s Manual for Tessent Shell* for a sample usage scenario.

Related Topics

delete_cdp_test

execute_cdp_test

add_cell_constraints

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

Mode: setup, analysis

Usage

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

Description

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

The add_cell_constraints command constrains scan cells slightly differently, depending on the tool. For the “patterns -scan” context, the command constrains a scan cell to a constant value during the ATPG process.

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

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

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

Note

Adding constraints can lower test coverage.

Table 7-3 on page 406 shows how cell constraints affect the cell during load-capture-unload. The table shows the unloaded (expected) value of the cell given three possible scenarios during capture:

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

- Cell Disturbed (different value captured): At least one of the cell's clock/set/reset ports is not off (either on or X), and the new stable capture value is different from the old value in the cell.

Note that the table uses ‘a’ and ‘b’ to represent different Boolean values (for example, if a=1, then b=0 or X). Each constraint is described in more detail in the “Arguments” on page 407.

Table 7-3. Example Cell Value Changes with Different Constraints for Scan Patterns

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

Table 7-4 shows, for chain patterns, the different load and unload values of a scan cell with different constraints applied. Notice for chain patterns that CX, XX and TX are equivalent, and DX and SX are the same as no constraint.

Table 7-4. Example Scan Cell Value Changes with Different Constraints for Chain Patterns

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

Arguments

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

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

- ***pin_pathname***

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

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

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

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

add_cell_constraints [get_gate_pins GO_ID_REG_reg_*/q -hierarchical] OX

- ***chain_name cell_position***

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

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

- **-CHain *chain_name* ("patterns -scan" context only)**

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

- **-CLock *clock_name* ("patterns -scan" context only)**

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

Note

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

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

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

- -Instances *instance_name* ...

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

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

  - If the state element is learned as TIE-0 or TIE-1, you cannot add any cell constraint.

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

- **C0**

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

- **C1**

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

- **CX**

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

- **OX**

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

- **XX**

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

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

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

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

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

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

- **SX ("patterns -scan" context only)**

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

Note

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

- **C0DX ("patterns -scan" context only)**

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

- **C1DX ("patterns -scan" context only)**

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

- -Scan_patterns **("patterns -scan" context only)**

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

- -All_patterns **("patterns -scan" context only)**

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

Note

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

Example 1

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

```
add_cell_constraints chain1 5 c0
add_cell_constraints /reg_d2_/Q ox -all_patterns
report_cell_constraints
```

```
// constraint chain   cell   cell_instance constraint    pattern scope
//    value   name  position      name     properties
// ---------- ----- -------- ------------- ---------- -------------------
//     CO     chain1    5                  (Dynamic)  Scan Patterns only
//     OX     chain4    7    /reg_d2_/Q    (Dynamic)  Chain + Scan Patt's
```

Example 2

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

add_cell_constraints -chain chain1 c1

Example 3

The following example solves C6 rule violations resulting from DRC:

add_cell_constraints -drc c6

Example 4

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

add_cell_constraints -clock /clk1 -type flop 0x

Related Commands

delete_cell_constraints
report_cell_constraints
report_scan_cells
report_scan_chains

add_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

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

Description

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

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

For more information on the design library attribute, see cell_type in the *Tessent Cell Library Manual*.

Note

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

Arguments

- ***dftlib_model***

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

- **-TYPE**

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

  INVerter — a literal specifying a 1-input inverter cell.

  AND — a literal specifying a 2-input AND cell.

  BUFfer — a literal specifying a 1-input buffer cell.

  OR — a literal specifying a 2-input OR cell.

  NAND — a literal specifying a 2-input NAND cell.

  NOR — a literal specifying a 2-input NOR cell.

  XOR — a literal specifying a 2-input exclusive OR cell.

CLOCK_BUFfer— a literal specifying a clock buffer cell.

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

SCAN_CELL ***clock data*** [-POSEDGE_clock | -NEGEDGE_clock] — a literal and two strings specifying one of the following scan cells:

- a mux-DFF scan cell with the following four input pins: clock, data, scan in, and scan enable
- a clocked-scan cell with the following four inputs: clock, data, scan clock, and scan enable
- a LSSD scan cell with the following five inputs: clock, data, scan in, master clock, and slave clock

You must specify the name of the clock and data pins of the DFT library cell model. You may also include an optional switch to specify the inversion on the clock pin. This option works in conjunction with the add_tied_signals and the set_registered_io commands.

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

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

Example

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

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

Related Commands

delete_cell_models
report_cell_models
set_lockup_cell
set_test_logic
set_registered_io

add_chain_masks

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

Mode: setup, analysis

Usage

Available when EDT is Off

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

Available when EDT is On

add_chain_masks ***chain_name...*** [-Cell_constraints {OX | XX | TX}]
[-Unload_value {X | 0| 1}]

Logic BIST simulation

add_chain_masks ***chain_name...*** [-Cell_constraints {OX | XX | TX}]
[-unload_value {0|1}] [-load_value {0|1}]

Description

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

The add_chain_masks command allows you to mask specified scan chain values from the test results. This command also allows you to specify a circuit response (0/1) to the compactor ("dft -edt" context) or output pins ("patterns -scan" context). In addition, this command allows you to load a value (0/1) on the (internal) scan chain inputs when you need to disconnect the scan chain inputs from the PRPG and avoid filling the scan chain with X's.

You can use this command to specify that custom masking logic exists between the scan chain outputs and the compactor ("dft -edt" context) or between the scan chain outputs and output pins ("patterns -scan" context). You must create and appropriately control the custom masking logic.

Note that this command does not impact DI fault classification. You can manually change the DI fault classification using the reset_di_faults command.

Logic BIST simulation

Specifies the scan chains that are masked during fault simulation and their load and unload values. Used to work around design issues. You must use this command to specify the chains you want masked.

Arguments

- ***chain_name...***

  Required, repeatable string that specifies the name of a scan chain in the current design.

- -Cell_constraints OX | XX | TX

  Optional switch and literal pair that specifies the cell constraints and observe values for the masking of the specified scan chain(s).

OX — Load and capture values are applied, and the fault observation is not included in the results.

XX — Load and unload values are not applied, and values are captured normally for all scan cells in the scan chain for multiple-cycle tests.

TX — All scan cell values are tied as X.

- -Unload_value X | 0 | 1

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

  X — Ties the unload value to X. This value is not valid for Logic BIST simulation.

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

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

- -Load_value 0 | 1

  Optional switch and literal pair that specifies the value that is applied to the (internal) scan chain input. You can use this switch only for designs that contain BIST logic. Options are:

  0 — Specifies the load value is 0.

  1 — Specifies the load value is 1.

Example 1

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

add_chain_masks chain1 -cell_constraints xx -unload_value 0

Example 2

This example shows the add_chain_masks in Context: pattern -scan. This command masks the scan chain input by loading a 0 and the scan chain output by unloading a 0 on chains chain1 and chain2. This data is transferred to pattern retargeting such that the chain mask register can be loaded with correct values to reflect this chain masking.

add_chain_masks chain1 chain2 -load 0 -unload 0

report_chain_masks

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

Related Command

delete_chain_masks

add_scan_chains

report_chain_masks

reset_di_faults

add_clocks

Context: all contexts

Mode: setup

Usage

For all contexts other than “patterns -ijtag” and “dft -scan”

add_clocks [*off_state*] ***pin_pathname…*** [-Internal [-PIn_name *user_pinname*]]
[{-Free_running [-PEriod *time_value* [s | ms | us | ns | ps | fs] [-Divider *int*]]}
| -Always_capture]

For “patterns –ijtag” (extraction mode inactive)

add_clocks [*off_state*] ***primary_input_port…***
[-Free_running [-PEriod *time_value* [s | ms | us | ns | ps | fs] [-Divider *int*]]]

For “patterns –ijtag” (extraction mode active)

add_clocks [*off_state*] ***pin_pathname…***
[-Free_running [-PEriod *time_value* [s | ms | us | ns | ps | fs] [-Divider *int*]]]
[-Internal [-PIn_name *user_pinname*] -REFERENCE_PIN *port*
[-REFERENCE_PIN_Inv *port*] [-FREQ_Divider *pos_int*] [-FREQ_Multiplier *pos_int*]]

For “dft -scan” and “dft -test_points”

add_clocks [*off_state*] ***pin_pathname...*** [-Internal [-Top_name *existing_pin*] -Inverted]
[-Free_running] | [-Always_capture]

Description

Adds scan or non-scan clock pins to the clock list for proper scan operation. Any signal is considered to be a clock if it can change the state of a sequential element, including system clocks, sets, and resets.

Pins that you add to the clock list default to an off-state of 0. The off-state of a clock pin is the value on the pin that results in the clock inputs of sequential memory elements becoming inactive. For edge-triggered devices, the off-state is the value on the pin that results in placing their clock inputs at the initial value of a capturing transition. Set and reset lines are also considered as clock lines. You can constrain a clock pin to its off-state in order to suppress its use as a capture clock during the ATPG process. The constrained value must be the same as the clock off-state or an error occurs. If you add an equivalence to the clock list, all of its equivalent pins are added to the clock list as well.

Arguments

- *off_state*

  A optional literal that specifies the pin value that inactivates the sequential memory elements. The choices are:

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

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

- ***pin_pathname***

  A required, repeatable string that typically lists the primary input pins to assign as clocks. The primary input pins in the list must have the same off_state. If the -Internal switch is used, pin_pathname lists internal pin pathnames. If both the -Internal and -Pin_name switches are used, pin_pathname lists internal pin pathnames to merge into a single, new primary input pin. The pin_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- ***primary_input_port***

  A required, repeatable string that lists the primary input ports to assign as clocks. The primary input ports in the list must have the same off_state.

- -Internal

  An optional switch that specifies that pin_pathname is an internal pin that, for ATPG purposes only, should be disconnected from its original driver and treated as if it were a clock primary input. If you use this switch, the pin_pathname argument must be an internal pin, not an actual clock primary input pin. When saving patterns, this internal clock input is not added to the top level interface. Use this switch to define internal clock inputs normally driven by on-chip circuitry.

Note

If you use the “write_patterns -Mode_internal” switch when saving serial or parallel simulation patterns, the test bench still forces the internal pins even though corresponding clock primary inputs are not included in the top-level interface. If you save the patterns using the -Mode_external switch, the test bench does not force those internal pins. In this case, unless the circuit generates values on the internal pins corresponding to what ATPG assumed was forced on the corresponding clock primary inputs, you may get simulation mismatches.

- -PIn_name *user_pinname*

  An optional switch and string pair that specifies the name for a new primary input clock pin that will drive all the internal pins specified with the pin_pathname argument. The user_pinname must be a single, simple name (wildcards are not allowed), which is given to the newly-created primary input pin. The -Pin_name switch is only allowed when the -Internal switch is also specified. The -Pin_name switch is also allowed if the pin_pathname argument specifies a single internal pin pathname and ensures a known pin name is used for the new primary input pin.

- -Free_running

  An optional switch that identifies the specified pin as a free-running clock. A free-running clock is pulsed during every cycle of every procedure. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting.

Free-running clocks are automatically pulsed in the external mode of named capture procedures (if present). Unless you define the free-running clock using the -Period switch, you must specify the clock in the internal mode of named capture procedures in order to simulate the effect of the clock pulses.

An always block is automatically created in the procedure file if it does not already exist, and the free-running clock is pulsed in this block. For more information, see "Always Block" in the *Tessent Cell Library Manual*.

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

  An optional switch and real number pair that you can use only with the -Free_running switch. The time_value is a real number followed by an optional time unit that defaults to "ns." This switch specifies that the free-running clock is an asynchronous clock with a certain time period. Note that ATPG and DRC treat asynchronous free-running clocks as X values.

  The period of an asynchronous free-running clock can affect the timescale and precision of the Verilog testbench. For more information, refer to the description of the "set time scale" statement in the *Tessent Shell User's Manual*.

- -Divider *int*

  An optional switch and integer pair that can only follow the -Period switch. When using the -Divider switch, the -Period switch specifies a base period and the -Divider switch specifies the number of pulses to occur during the base period. The -Period switch and -Divider switch are intended for coherent clock groups, to specify the base period for a number of clock pulses and the total pulses during that base period. All clocks that share the same base period are grouped together as coherent clocks in the final Verilog testbench. For more information about using these two switches, refer to "Example 5" on page 421.

- -Always_capture

  An optional switch that identifies the specified pin as an always-capture clock. An always-capture clock is pulsed in every cycle of the capture window, except when using named capture procedures. The clock you specify also pulses with any other clock, ignoring the specified clock restriction setting.

- -REFERENCE_PIN *port* **(patterns -ijtag only)**

  Switch and string pair that is required when you specify the -internal switch in ICL extraction mode. The *port* argument specifies the source primary input port that is connected to the internal clock pin you specified with the *pin_pathname* argument.

- -REFERENCE_PIN_Inv *port* **(patterns -ijtag only)**

  An optional switch and string pair that specifies the second primary input port if the reference_pin is a differential pin.

- -FREQ_Divider *pos_int* **(patterns -ijtag only)**

  An optional switch and positive integer pair that defines the division factor of the clock frequency between the reference pin and the internal clock pin specified with the *pin_pathname* argument.

- -FREQ_Multiplier *pos_int* **(patterns -ijtag only)**

  An optional switch and positive integer pair that defines the multiply factor of the clock frequency between the reference pin and the internal clock pin specified with the *pin_pathname* argument.

- -Top_name *existing_pin* **(dft -scan only)**

  An optional switch and string pair that specifies the name of an existing top level pin that drives the internal node during scan chain shifting. The top level pin should be a clock signal; you do not need to define it using the add_clocks command. Note: Although the top_name switch is optional, the tool issues a warning message if it cannot automatically trace from the internal node to a primary input pin (the pin must not be a scan signal, and must not have any pin constraints).

- -Inverted **(dft -scan only)**

  An optional switch that must be used in conjunction with the -top_name switch to indicate an inverting path; this may be necessary if the signals cannot be traced due to a black-boxed module.

Example 1

The following example adds a clock to the clock list with an off-state of one, then adds an internal clock with an off-state of zero:

add_clocks 1 clock1
add_clocks 0 pll_block/clock1 -internal
report_clocks

Example 2

The following example adds an internal clock pin to a net, disconnecting the original driver of the net. This happens within the tool only; the tool does not modify the original source netlist. Then the example reports on the added clock pin and the original driver of the net. The example begins in Setup mode.

report_clocks

```
User-defined Clock (1):
 ---------  -----------  --------  ----  ----------------
 Off State  Constraints  Internal  Name  Other Properties
 ---------  -----------  --------  ----  ----------------
     0                      No     /clk
```

create_flat_model
report_gates /reg_d_1_7_/Q

```
//  /reg_d_1_7_   dff
//      CLK    I  /clk
...
//      Q      O  /ix225/A1
```

add_clocks 1 /ix225/A1 -internal

```
//  Warning: Flattened model deleted.
```

report_clocks

```
User-defined Clock (1):
 ---------  -----------  --------  ----  ----------------  ------
 Off State  Constraints  Internal  Name  Other Properties  Pin(s)
 ---------  -----------  --------  ----  ----------------  ------
     0                      No     /clk
     1                      Yes    /d_1_7_                 /ix225/A1
```

report_primary_inputs

```
USER:  /d_1_7_ (internal pin)
...
SYSTEM:  /clk
...
```

set_system_mode analysis
report_gates /d_1_7_

```
//  /d_1_7_  primary_input
//      d_1_7_  O  /ix225/A1
//
//  User-added pin driven by "/reg_d_1_7_/Q" in the source netlist.
```

report_gates /reg_d_1_7_/Q

```
//  /reg_d_1_7_  dff
//      CLK    I  /clk
...
//      Q      O
...
//
//  Gate drives user-added pin "/d_1_7_" in the source netlist.
```

Example 3

The following example adds a list of pathnames of internal clock pins to connect to the new primary input pin “div_pen” with an off-state of zero. An asterisk wildcard character is used in the example to only add pins with pathnames that end with divider_gate/CPEN.

add_clocks 0 *divider_gate/CPEN -internal -pin_name div_pen

Example 4

The following example specifies a free running clock named “sclk” with an off state of 0 and a total period of 40ns with a 50% duty cycle. In the Verilog testbench, the clock starts at time 0 in its off state:

add_clocks 0 sclk -free_running -period 40

Example 5

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

add_clocks aclk -free_running -period 1000 ns -divider 20
add_clocks bclk -free_running -period 1000 ns -divider 30

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

```
{  Begin_async_clocks }
{    sclk =  period: 100 ps  }
{  End_async_clocks }
```

If a timeplate name is used to specify the timing for the asynchronous clock, then additional information on the leading and trailing edge is also specified. If the -Divider switch is used, this is also present in the annotations:

```
{  Begin_async_clocks }
{    sclk = period: 100ps, leading: 40ps, trailing: 70ps }
{    aclk = period: 1000ns, divider: 20 }
{  End_asyn_clocks }
```

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

Example 6

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

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

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

```
Module chip {
   // Created by ICL extraction
   ClockPort clock1;
   …
   Instance block1_I1 Of block1 {
      InputPort ClkA = add_clock_inst_1.ClockOut;
       …
   }
   Instance add_clock_inst_0 Of chip_add_clock_0 {
      InputPort ClockIn = ClkA;
   }
}

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

```
        }
    }
```

Related Commands

analyze_control_signals
delete_clocks
report_clocks
set_clock_restriction

add_config_element

Context: all contexts including unspecified

Mode: all modes

Usage

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

Description

Adds a configuration element in the configuration tree.

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

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

Arguments

- *name*

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

- -copy_from *config_object_spec*

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

copy_from and the -in_wrapper options. In this case, the parent wrapper of the copied element and the destination wrapper must be different. The element is added into the destination wrapper using the leaf_name of the copied object when the *name* option is not specified.

- -in_wrapper *wrapper_object_spec*

  An option value pair that specifies that the configuration element is to be added inside a specific wrapper. When the -in_wrapper option specified, the name option defines a name relative to the specified destination wrapper. The wrapper_object_spec can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by get_config_elements or "get_config_value -object".

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

- -before *config_object_spec*

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

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

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

- -after *config_object_spec*

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

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

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

- -first

  An optional option that specifies to add the new configuration element before any other element in the parent wrapper.

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

- -last

  An optional option that specifies to add the new configuration element after all other elements in the parent wrapper.

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

- -type wrapper | data_wrapper | csv_wrapper | property

  An optional option value pair that is only needed and allowed when adding a configuration element for which no Wrapper exists. When adding an element for which metadata exists, the -type option is not allowed because the type of the object is specified in the metadata. When adding an element for which no metadata exists, the -type value defaults to “wrapper” when the -value option is not specified and to “property” when the -value option is specified.

  See Example 2 for an example of its usage when adding an element without metadata.

- -value *value_list*

  An optional option value pair that is used when adding a configuration element that needs a value. The -value option is typically needed when -type is specified as a property.

- -replace

  An optional option that suppresses the error that is normally generated when adding a configuration element with a name that conflicts with an existing element. The error only occurs if the element is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Example1

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

```
> set_transcript_style input_only
> set spec    [add_config_element DftSpecification(moda,ex1)]
> set ijtag   [add_config_element IjtagNetwork -in $spec]
> set host    [add_config_element HostScanInterface(ijtag) -in $ijtag]
> set mbist   [add_config_element MemoryBist -in $spec]
> set_config_value ijtag_host_node -in $mbist HostScanInterface(ijtag)
> set ctl     [add_config_element Controller(c1)  -in $mbist]
> set step    [add_config_element Step -in $ctl]
> set mem_int [add_config_element MemoryInterface(m1) -in $step]
> set_config_value memory_library_name -in $mem_int ram12x4
> set mem_int [add_config_element ReusedMemoryInterface(m2) -in $step]
> set_config_value reused_interface_id -in $mem_int m1
> set step    [add_config_element  Step -in $ctl]
```

```
> set mem_int [add_config_element MemoryInterface(m3) -in $step]
> catch {set_config_value memory_library_name -in $mem_int ram12x8}
> set mem_int [add_config_element ReusedMemoryInterface(m4) -in $step]
> set_config_value reused_interface_id -in $mem_int m3
> report_config_data DftSpecification(moda,ex1)

DftSpecification(moda,ex1) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
    }
  }
  MemoryBist {
    ijtag_host_node : HostScanInterface(ijtag);
    Controller(c1) {
      Step {
        MemoryInterface(m1) {
          memory_library_name : ram12x4;
        }
        ReusedMemoryInterface(m2) {
          reused_interface_id : m1;
        }
      }
      Step {
       MemoryInterface(m3) {
          memory_library_name : ram12x8;
        }
        ReusedMemoryInterface(m4) {
          reused_interface_id : m3;
        }
      }
    }
  }
}
```

Example 2

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

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

tmp(1) {
```

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

Related Commands

add_config_element
delete_config_elements
get_config_elements
move_config_element

add_config_message

Context: all contexts

Mode: all modes

Usage

add_config_message **-config_object** ***config_object message***
[-error | -warning | -info]
[-display]

Description

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

Arguments

- **-config_object** ***config_object message***

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

- -error

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

- -warning

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

- -info

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

- -display

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

Example

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

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

Related Commands

delete_config_messages

get_config_messages

report_config_messages

add_config_tab

Context: all contexts

Mode: all modes

Usage

add_config_tab ***wrapper***

Description

Adds one or many configuration tree tabs in the DFTVisualizer Configuration Data window.

Arguments

- *wrapper*

  An optional value that specifies the name of the configuration wrapper. The value is either the name of a wrapper, a Tcl list of names of wrappers, or a collection containing one or more configuration wrappers. The wrappers are the root wrapper displayed in the tab.

Example

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

```
> add_config_tabs [get_config_element Core(RAM12x8) -partition tcd]
```

Related Commands

delete_config_tabs

display_specification

add_control_points

Context: dft -test_points

Mode: setup, analysis

Usage

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

Description

Specifies user-defined control points during test point insertion.

Arguments

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

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

  The ***pin/port_spec*** can be the name of a pin/port, a Tcl list of pins/ports, or a collection of pin/port objects.

- **AND | OR**

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

- -scan_cell_clock ***pin/port_spec***

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

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

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

Example 1

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

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

Example 2

The following example specifies the path name of an existing net that is to be used to enable/disable the control points. For the AND control point it also specifies the path name of

the clock signal used to clock the control point. Then it proceeds to insert the control points into the netlist:

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

Related Commands

add_observe_points

delete_test_points

report_test_points

add_core_instances

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Usage

add_core_instances **-core *core_name*** [-mode *mode_name*]
{**-modules *module_objects*** | **-instances *instance_objects*** | **-current_design**}
[-silent]

Description

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

The read_core_descriptions command must have already been executed before using this command.

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

Arguments

- **-core *core_name***

  A required switch and Tcl list that specifies the core name of the core description from which the instantiation should take place.

- -mode *mode_name*

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

- -modules *module_objects*

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

- -instances *instance_objects*

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

- -current_design

  A required switch that specifies to add the core instance with respect to the current design. This switch is mutually exclusive with the -modules and -instances switches. The mode of the core instance being added must be different than the mode of the current design.

- -silent

  An optional switch that specifies to suppress warning messages.

Example 1

The "Retargeting Example" in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Example 2

The following example demonstrates the use of the -current_design switch to merge two modes of a core; in this case, the "core" is the top level. In the example, patterns are generated separately for two sub-blocks of a design from the top level pins. Sub-Block1 is tested as mode M1 of the top level, and Sub-Block2 is tested as mode M2 of the top level. The result, shown in Figure 7-1, is the merging of the two pattern sets originally generated for each individual mode into a single pattern set applied at the top level.

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

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

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

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

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

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

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

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

Figure 7-1. Merging Patterns for Multiple Views of a Core at the Top Level

Example 3

The following example demonstrates the use of the add_core_instances -current_design switch to merge mode M1 of the current design and mode internal of CoreB. In the example, one pattern set is generated at the top level (mode M1 of the top level) to test Sub-Block1, and a second pattern set is generated at the core level of CoreB. The result, as shown in Figure 7-2, is the integration of the two pattern sets, originally generated for mode M1 of the top level and mode internal of CoreB, into the top level.

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

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

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

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

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

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

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

Figure 7-2. Merging Patterns for Top Level and Core at the Top Level

Related Commands

delete_core_instances
read_core_descriptions
report_core_instances
write_core_description

add_display_data

Context: all contexts

Mode: setup, analysis

Usage

add_display_data {
Error_pattern | **PATtern_index** ***pattern_index*** [-Internal | -External] |
Parallel_pattern ***pattern_number*** (0..31)| **Fault_status** | **TIe_value** |
COnstrain_value | **Seq_depth_data** | **CLock_cone** ***pin_name*** |
CAPTURE_PROCedure ***procedure_name*** |
Drc_pattern {{**Test_setup** [{{-Cycle | -Time} *n1*} [*n2* | End]]} | **Load_unload** | **SHIft** |
SKew_load | **SHADOW_Control** | **Master_observe** |
SHADOW_Observe | **STate_stability** | **TEST_End [***time* | -All**]**}}

Note

The following DFTVisualizer menu paths are available when the associated data or procedure is available; so you may not see all these selections all the time.

Possible DFTVisualizer Menu Paths:
Data > Capture Procedure | Test-Setup | Load-Unload | Shift | State-Stability | Error Pattern | Pattern Index | Parallel Pattern | Fault | Tied | Constrain | Sequential Depth | Clock Cone | Test-End

Note

The following arguments are not supported in dft -scan: Pattern_index, Parallel_pattern, Fault_status, and Seq_depth_data.

Description

Adds specified data columns to the Data window of DFTVisualizer, opening the Data window if it is not already open. If data corresponding to an added column is available, it is displayed.

The command's arguments and usage are based on the set_gate_report command and control the types of data displayed similar to how the set_gate_report command controls the data output of the report_gates command.

If a column has been added and then removed during a tool session, you can re-add it by moving the cursor to the column title bar, then using the right mouse button popup menu to reselect the column for display.

Use the delete_display_data command (or the right mouse button popup menu) to remove data columns you no longer need.

Arguments

For complete detailed descriptions of the type of data that the arguments control, refer to the similar arguments listed in the set_gate_report command description.

Examples

The following example adds a column displaying the simulated values for cycles 5 through 10 of the test_setup procedure:

```
add_display_data drc_pattern test_setup -cycle 5 10
```

Related Commands

delete_display_data
report_display_instances
report_gates
set_gate_report

add_display_instances

Context: all contexts

Mode: setup, analysis

Usage

add_display_instances {{***gate_id#*** [-Pin *pin_name*] | ***pin_pathname*** | ***instance_name***
[-Pin *pin_name*] | **-PAth** ***start end***}}...}
[-Backward [-One | -End_point] | -Forward [-One | -All | -End_point] | -Up | -DOwn]
[-DIsplay {DEBug | DESign | DAta | WAve | HIerarchy | LIbrary | ALl}]

Description

Displays instances in DFTVisualizer and enables you to trace visually through the design using the Debug or Design window.

The add_display_instances command has two purposes:

- Adds display instances to DFTVisualizer windows

  How DFTVisualizer updates the display in the Debug window when you add an instance depends on the gate level that is in effect, as summarized in the following table:

  Table 7-5. How the DFTVisualizer Debug Window Displays Instances

Gate Level of Instance You Specify	"set_gate_level Design" in effect	"set_gate_level Primitive" in effect
Primitive	Displays the design level instance of which the primitive is a part	Displays the primitive
Design	Displays the design level instance	Displays all the primitives in the design level instance

- Traces visually through the design

  Tracing is the process of displaying additional gates that are connected to an already displayed gate. You might want to trace in order to expand the schematic view, gain insight into circuit behavior, or get oriented within a large design. In the Debug window, you can trace from a displayed instance forward towards the primary output pins or backward towards the primary input pins at the gate level currently specified by the set_gate_level command. In the Design window, you can additionally trace from a displayed instance up or down the design hierarchy.

Arguments

The following arguments describe the methods for naming the instances you want DFTVisualizer to display. You can specify any number of these five arguments, in any order.

- ***gate_id#***

  A repeatable integer and optional switch and string pair that specifies a gate to display in DFTVisualizer. The value of the *gate_id#* argument is the unique identification number the tool automatically assigns to every gate within the design during the model flattening process.

- ***pin_pathname***

  A repeatable string that specifies the pin pathname (<instance pathname>/<pinname>) of a pin within the design. DFTVisualizer displays the gate associated with that pin.

- ***instance_name***

  A repeatable string that specifies the instance pathname of an instance within the design. DFTVisualizer displays the gate associated with that instance. You can include any number of asterisks (*) or question marks(?) as wildcard characters to specify this string enabling you to match many pathnames in the design.

Note

For the Design and Data windows, *instance_name* can also be a hierarchical submodule. For the Debug window, *instance_name* must be a design level instance.

- **-Path** ***start end***

  A repeatable switch and two-string triplet that specifies to display all gates in the circuit path beginning at the *start* gate and ending at the *end* gate. Start and end can be a *gate_id#* or *pin_pathname*.

Note

When this switch is specified in the Design window, the tool attempts to trace the path in one direction beginning at *start* gate; the tool returns an error if *end* gate is not found. In the Debug window, the tool first attempts to trace the path in the direction specified by the user. If that path is not found, the tool then attempts to trace the path in the opposite direction. If that path in not found, the tool returns an error message.

- -Pin *pin_name*

  An optional switch and string pair that specifies a particular input or output pin of the specified gate or instance.

The next four switches determine the direction you want the tool to trace from the specified gate(s):

Note

The trace switches are not supported in the Data window.

- -Backward

  An optional switch that specifies to trace from the given instances backward, towards the primary input pins.

  If you do not explicitly specify a stopping point switch (-One, -End_point) in combination with this switch, the command default is for the backward trace to include only *one* level of gates (Debug window) or one hierarchal instance (Design window).

- -Forward

  An optional switch that specifies to trace from the given instances forward, towards the primary output pins.

  If you do not explicitly specify a stopping point switch (-One, -End_point, or -All) in combination with this switch, the command default is for the forward trace to include only *one* level of gates (Debug window) or one hierarchical instance (Design window).

- -Up

  An optional switch that specifies for DFTVisualizer to trace up one level of hierarchy from the given instance(s). This switch is not valid for the Debug window.

- -DOwn

  An optional switch that specifies for DFTVisualizer to trace down one level of hierarchy from the given instance(s). This switch is not valid for the Debug window.

Each of the following optional switches determine the stopping point for a trace by specifying the last gate you want DFTVisualizer to include in the display. If used, select just one to include in the command string.

- -One

  An optional switch whose behavior depends on whether you are doing a backward trace or a forward trace, as follows:

  Backward trace — Display only the first gate backward from a specified input pin (or from each input pin of a specified instance) from which you are tracing. This is the default for a backward trace.

  Forward trace — Display only one fanout gate connected to a specified output pin (or connected to each output pin of a specified instance) from which you are tracing, even if the pin fans out to multiple gates.

  This switch must be accompanied by the -Forward or -Backward switch.

- -End_point

  An optional switch that specifies to continue the trace until reaching either a primary input, primary output, tie gate, memory, or sequential element (latch or flip-flop, scan or nonscan). This switch must be accompanied by the -Forward or -Backward switch.

  This switch currently is not supported in the Design window.

- -All

  An optional switch that, together with the -Forward switch, specifies to trace to all the fanout gates connected to an output pin from which you are tracing. This switch must be accompanied by the -Forward switch.

- -Display *window_name*

  An optional switch and repeatable literal that specify the DFTVisualizer windows to which the tool should add instances and data sets if available. The valid options for the *window_name* argument, from which you can select as many as you want, are as follows:

  DEBug
  A literal that specifies the Debug window.

  DESign
  A literal that specifies the Design window.

  DAta
  A literal that specifies the Data window.

  WAve
  A literal that specifies the Wave window.

  HIerarchy
  A literal that specifies the Hierarchy window.

  LIbrary
  A literal that specifies the Library window.

  ALl
  A literal that specifies all the preceding windows.

  The default is to add instances to the Debug window only.

Examples

The following example opens DFTVisualizer if not already open and displays a single gate in the Debug window by specifying the gate identification number (51):

```
add_display_instances 51
```

The next example specifies for the tool to additionally display all the gates in the first level of fanout from the Q output of gate 51. Gates connected directly to the Q output comprise the first level of fanout gates. (Gates fed by the first level of fanout gates comprise the second level of fanout gates and are not displayed.)

```
add_display_instances 51 -pin Q -forward -all
```

The next example clears the schematic display of all gates, then creates a new display that shows the specified gate, along with a backtracking of all the gates until the trace reaches either a primary input or tie gate.

```
delete_display_instances -all
add_display_instances 51 -backward -end_point
```

Related Commands

delete_display_instances
read_modelfile
report_display_instances
set_gate_level

add_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

add_edt_blocks ***block_name***

Description

Creates an arbitrary identifier for an EDT block and applies certain subsequent commands only to that block.

This command defines a string, referred to as the block name or block tag, as the identifier for an EDT block instantiated in a design netlist and simultaneously restricts the applicability of certain subsequent commands to that block. Restricting the applicability of these commands to a particular EDT block is referred to as setting or changing the EDT context, and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command or change the context to a different EDT block using set_current_edt_block or another add_edt_blocks command.

You must use this command to define a unique name for an EDT block prior to defining its attributes using context-sensitive commands. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (set_system_mode, create_patterns and report_statistics for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block ("set_edt_options -channels" for example) returns an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- ***block_name***

  A required string that specifies a name to use for uniquely identifying a specific EDT block instantiated within a netlist. You can use any unique name you like, as it is independent of the module and instance names used in the netlist.

Example 1

The following example defines the block name, "my_core1_block" for an EDT block that contains four scan chains. The tool determines the EDT block to associate with this name from

the pin pathnames specified in the add_scan_chains commands that follow the add_edt_blocks command.

add_edt_blocks my_core1_block

```
# my_core1_block is the current EDT block.
```
...

add_scan_chains -internal core1_chain1 grp1 /edt_top_core1/cpu1_i/edt_si1
/edt_top_core1/cpu1_i/edt_so1
add_scan_chains -internal core1_chain2 grp1 /edt_top_core1/cpu1_i/edt_si2
/edt_top_core1/cpu1_i/edt_so2
add_scan_chains -internal core1_chain3 grp1 /edt_top_core1/cpu1_i/edt_si3
/edt_top_core1/cpu1_i/edt_so3
add_scan_chains -internal core1_chain4 grp1 /edt_top_core1/cpu1_i/edt_si4
/edt_top_core1/cpu1_i/edt_so4

Example 2

The following example defines two more block names, “my_core2_block” and “my_core3_block”, then displays a list of the current block names. The display indicates subsequent context-sensitive commands will apply only to the EDT block named “my_core3_block”.

add_edt_blocks my_core2_block

```
# my_core2_block is the current EDT block.
```
...
add_scan_chains -internal core2_chain1 grp1 /edt_top_core2/cpu3_i/edt_si1
/edt_top_core2/cpu3_i/edt_so1
add_scan_chains -internal core2_chain2 grp1 /edt_top_core2/cpu3_i/edt_si2
/edt_top_core2/cpu3_i/edt_so2
...

add_edt_blocks my_core3_block

```
# my_core3_block is the current EDT block.
```
...
add_scan_chains -internal core3_chain1 grp1 /edt_top_core3/cpu7_i/edt_si1
/edt_top_core3/cpu7_i/edt_so1
...
add_scan_chains -internal core3_chain8 grp1 /edt_top_core3/cpu7_i/edt_si8
/edt_top_core3/cpu7_i/edt_so8

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
//  my_core3_block   (current block)
```

Example 3

The following example removes from the tool's internal database, the data for the EDT block named "my_core3_block", making it unavailable for use with the set_current_edt_block command.

delete_edt_blocks my_core3_block

```
// Block deleted. No EDT block now selected as current block.
```

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
```

Related Commands

delete_edt_blocks
report_edt_blocks
set_current_edt_block

add_edt_configurations

Context: dft -edt

Mode: setup

Usage

add_edt_configurations ***config_name***

Description

Sets up EDT logic with two compression configurations.

The add_edt_configurations command creates a compression configuration and applies subsequent EDT channel specifications to the configuration. Up to two compression configurations can be defined for a design block.

In a multi-block design flow, you can define the compression configurations at the top-level integration and all subsequently added blocks inherit the defined compression configurations. The configuration parameters need to be set for each block context.

Note

When two configurations are specified and one of the configurations is missing an input/output channel parameter, it inherits the needed value from the other configuration. You can override this feature by explicitly specifying the channel configuration.

For more information on setting up two compression configurations, see the "Defining Dual Compression Configurations" section in the *Tessent TestKompress User's Manual*.

Arguments

- ***config_name***

  A required string that specifies a name for the compression configuration.

Example

The following example shows a dofile that creates two configurations for a single block.

```
# edt_ip_creation.do
#
# Dofile for EDT logic creation
dofile scan_chain_setup.dofile

# Set up EDT configurations
set_edt_options -location internal
add_edt_configurations my_sys_test_config
set_edt_options -input_channels 4 -output_channels 2
add_edt_configurations my_wafer_test_config
set_edt_options -input_channels 5 -output_channels 4

# Set bypass pin
set_edt_pins bypass my_bypass_pin
```

```
# Set edt configuration pin
set_edt_pins configuration my_configuration_pin
set_system_mode analysis

# Report and write EDT logic.
report_edt_configurations -all # Report configurations for all blocks.
report_edt_pins # Report all pins including compression configuration
# specific pins.
write_edt_files created -verilog -replace # Create dofiles and
# test procedure files for both the configurations and the bypass mode
write_design -output_file created_edt_top_rtl.v -replace # Write the
# current design in Verilog format to the file created_edt_top_rtl.v.
```

Related Commands

delete_edt_configurations
report_edt_configurations
set_current_edt_configuration

add_edt_connections

Context: dft -edt

Mode: setup

Usage

add_edt_connections {[-Signal ***type*** {-To *connection_pathname* | -No_connection}] [-Blocks {*block_name*...} | -All_blocks]} [{-From *connection_pathname*} {-To {*connection_pathname*...}}]

Description

Specifies EDT logic connections for the block-level design flow.

Use this command to specify the connections needed during the integration of EDT logic blocks into the top-level netlist as follows:

- Top-level connections for the EDT control signals
- Other connections between pins/nodes in the design

For information, see "Creating the Top-level Netlist" in the *Tessent TestKompress User's Manual*.

If the *-signal* switch is specified as *scan_enable* or *-signal* is not specified at all, the tool uses simulation to virtually resolve the stitching of a pin at the top level to the block level by identifying a source and sink node. Typically, the source node is an internal connection node, and the sink node is the block port that corresponds to the block-level signal.

When test procedures are simulated, the events on the top-level signals are simulated and the event propagates to the internal connection node (source). (Note that this event propagation path may have inverters/buffers.) When an event is detected on the internal connection node (source), a corresponding event is scheduled on the sink node; this mimics a virtual wire connection between the source and sink that is not present during the integration phase but will be present during the pattern generation phase when EDT connections are physically materialized.

For example, a T3 DRC violation would normally occur when stitching the scan enable at the top level because the scan enable would only be stitched after DRC. Simulation avoids the T3 violation by allowing the block-level scan enable port to be driven correctly and scan chain tracing will pass.

Arguments

- -Signal ***type***

  An optional switch and literal pair that specifies an EDT control signal. Valid literals include:

  clock — specifies the edt_clock pin.

reset — specifies the edt_reset pin.

scan_enable — specifies the top-level scan_en pin.

bypass — specifies the edt_bypass pin.

update — specifies the edt_update pin.

configuration — specifies the edt_configuration pin for dual configuration applications.

master_clock — specifies the master_clock pin. Use in latch-based EDT logic for LSSD designs only.

slave_clock — specifies the slave_clock pin. Use in latch-based EDT logic for LSSD designs only.

input_channel *N* — appended with the channel number to specify an EDT input channel.

output_channel *N* — appended with the channel number to specify an EDT output channel.

single_bypass_chain — edt_single_bypass_chain pin.

low_power_shift_en — low power shift enable pin for low-power decompressor.

- -To *connection_pathname...*

  Optional switch and repeatable string pair that specifies the pathname of the destination pin/node for a connection. The connection_pathname can be an existing top-level pin, a new top-level pin, or an internal node. Multiple destination pins can be specified for non-EDT pin connections.

- -From *connection_pathname*

  Optional switch and string pair that specifies the pin pathname of the source pin/node for a non-EDT connection. The connection_pathname can be an existing top-level pin, a new top-level pin or an internal node.

- -No_connection

  Optional switch that specifies no connection for an EDT pin.

- -Blocks *block_name***...**

  Optional switch and repeatable string pair that specifies one or more blocks. If no blocks are specified, the current one is assumed.

- -All_blocks

  Optional switch that specifies all blocks. If no blocks are specified, the current one is assumed.

Example

The following example integrates two blocks, B1 and B2, and connects a non-EDT pin to test_en.

Add top-level clocks/pin constraints and test procedure file:

```
add_scan_groups grp1 top.testproc
add_clocks 0 clk1 clk2
```

Turn on EDT mapping to enable reading block level dofiles:

```
set_edt_mapping on
```

Add the EDT blocks:

```
add_edt_blocks B1
set_edt_instances -block_location B1_inst
dofile B1_edt.dofile
add_edt_blocks B2
set_edt_instances -block_location B2_inst
dofile B2_edt.dofile
```

Specify connections to block-level EDT signals:

```
add_edt_connections -signal input_channel 1 -to TK_channel_in[0] -block B1
add_edt_connections -signal output_channel 1 -to TK_channel_out[0] -block B1
add_edt_connections -signal input 1 -to TK_channel_in[1] -block B2
add_edt_connections -signal output 1 -to TK_channel_out[1] -block B2
add_edt_connections -signal clock -to TK_clk -blocks B1 B2
add_edt_connections -signal update -to TK_upd -all_blocks
add_edt_connections -signal bypass -to /tap_i/tap_ctrl_i/edt_bypass_inst -all_blocks
```

Optionally, specify a connection for a non-EDT signal:

```
add_edt_connections -from tap_i/tap_ctrl_i/test_en \
-to /B1_inst/test_en /B2_inst/test_en
```

Finally, change to analysis mode and make the specified pin connections:

```
set_system_mode analysis
report_edt_connections -all_blocks
write_edt_files integrate -replace
write_design -output_file integrate_edt_top_rtl.v -replace
```

Related Commands

delete_edt_connections

report_edt_connections

add_false_paths

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

Mode: setup, analysis

Usage

add_false_paths
{[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]} | **-CRoss_clock_domains**
[-Hold] [-Setup]

Description

Defines false paths in the design. False paths are paths that cannot be activated in the design's functional (at-speed) mode of operation. You would typically determine them as a part of static timing analysis, prior to ATPG.

If a false path is sensitized for an at-speed test during ATPG, incorrect capture responses can occur at certain scan cells and POs with resulting incorrect capture responses at the end points of the false path. This can cause simulation mismatches when you verify the patterns in a timing-based simulator. By defining the false paths, you enable the tool to identify the incorrect capture responses. Once identified, the tool will mask these responses (modify them to X) in the resultant patterns so they will not be the source of simulation mismatches.

For additional information, refer to "Preventing Pattern Failures Due to Timing Exception Paths," in the *Tessent Scan and ATPG User's Manual*.

Arguments

Tip: When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified source node does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

  If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified sink node does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database.

Note

For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “add_false_paths... -to CLK2” would define paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

  add_false_paths -from A -through B C -through D E -to F

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool will consider all paths from the specified starting points to the specified end points to be false paths.

If a through node does not translate to an internal gate, the tool will transcript an error message and not add the false path definition to its internal database.

- **-CRoss_clock_domains**

  A switch that adds hold time false paths between all clocks.

Note

You cannot use this switch in Setup mode.

Use this switch instead of multiple add_false_paths command instances using the -TO_Clock and -FROM_Clock switches to specify each path and direction between the clocks. See Example 7.

- -Hold

  An optional switch that evaluates the false path for hold time violations.

- -Setup

  An optional switch that evaluates the false path for setup time violations.

  For more information, see "Types of Timing Exception Path Violations" in the *Tessent Scan and ATPG User's Manual*.

Example 1

The following example creates three false paths definitions, then reports all current false path definitions:

```
add_false_paths -from  /my_design/a  /my_design/b  -through  /my_design/u1
add_false_paths -to  /my_design/c
add_false_paths -through  /my_design/u24  /my_design/u25
report_false_paths -all
```

```
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3
```

Example 2

The following example defines the path highlighted in Figure 7-3 as a false path:

```
add_false_paths -from  DFF1/q  -to  DFF5/d
```

Figure 7-3. Example of a False Path to be Defined

Here are four more ways you could define the same false path:

add_false_paths -from DFF1/q -through NOR1 -to_clock CLK

add_false_paths -from DFF1/clk -to DFF5/d

add_false_paths -from DFF1/q -to DFF5/clk

add_false_paths -from DFF1/clk -to DFF5/clk

Example 3

Referring to Figure 7-4, the following example defines as false paths all the paths that start from Reg_A[0...7] and end at Reg_B[0...3]:

add_false_paths -from Reg_A -to Reg_B

Figure 7-4. Example Circuit for Defining False Paths

Example 4

This example defines as false paths (in Figure 7-4) all the paths that start from Reg_A[0] and end at Reg_B[0...3]:

add_false_paths -from Reg_A[0] -to Reg_B

Example 5

This example defines as false paths all the paths that start from Reg_A[0] or Reg_A[3] and end at Reg_B[2]:

add_false_paths -from Reg_A[0] Reg_A[3] -to Reg_B[2]

Example 6

The remaining six examples describe signal matches for various sample pin pathnames specified using the asterisk (*) wildcard character. The pathnames are based on the following design:

```
module top(...);
   Input ...;
   Output...;
   ...
   foo u1(...);
   ...
endmodule;

module foo(...);
   Input...;
   Output...;
   ...
   staticRegister I_reg_1(.D1(...), .C1(...), .Z(...));
   staticRegister I_reg_2(.D1(...), .C1(...), .Z(...));
   staticRegister I_reg_3(.D1(...), .C1(...), .Z(...));
   staticRegister F_reg_1(.D1(...), .C1(...), .Z(...));
   staticRegister F_reg_2(.D1(...), .C1(...), .Z(...));
```

```
    ...
endmodule;
```

1. The signal name /u1/*reg_1/D1 will map to signals /u1/I_reg_1/D1 and /u1/F_reg_1/D1.
2. The signal name /u1/I_reg*/D1 will mapto signals /u1/I_reg_1/D1, /u1/I_reg_2/D1 and /u1/I_reg_3/D1.
3. The signal name /u1/*reg*/Z will map to signals /u1/I_reg_1/Z, /u1/I_reg_2/Z, /u1/I_reg_3/Z, /u1/F_reg_1/Z and /u1/F_reg_2/Z.
4. The signal name /u1/F_reg_2/* will map to signals /u1/F_reg_2/D1, /u1/F_reg_2/C1 and /u1/F_reg_2/Z.
5. The signal name /u1/F*/*1 will map to signals /u1/F_reg_1/D1, /u1/F_reg_1/C1, /u1/F_reg_2/D1 and /u1/F_reg_2/C1.
6. The signal name /u1/I*1/C1 will map to signal /u1/I_reg_1/C1.

Example 7

Assume the design contains two defined clocks: clk1 and clk2. The following command adds hold time false paths between clk1 and clk2:

add_false_paths -cross_clock_domains -hold

The above command is equivalent to the following two commands:

add_false_paths -from_clock clk1 -to_clock clk2 -hold
add_false_paths -from_clock clk2 -to_clock clk1 -hold

Related Commands

delete_false_paths
delete_multicycle_paths
read_sdc
report_false_paths
report_multicycle_paths

add_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Usage

add_fault_sites {**-ALl** | **-UNDEFINED_Cells**} [-VERBose]

Description

Automatically adds fault site definitions for library cells that don't have UDFM definitions.

The add_fault_sites command automatically adds port fault site definitions for library cells that are not covered by the current UDFM definitions so that you don't have to add them manually to the UDFM file. In order to distinguish those added definitions and resulting fault sites in report functions, the added definitions are collected in the UDFM type "auto-inserted-faults."

After you've added the fault sites, the add_faults command adds static or delay faults to those locations, as specified by the "set_fault_type udfm" command.

Note

The add_fault_sites command works only for the fault type UDFM, and is available only after you read a UDFM fault file and before you create the fault list.

For more information about UDFM, refer to "User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-ALl**

  A required switch that automatically adds UDFM definitions for all undefined (uncovered) port fault sites, including PI's and PO's. The fault sites affected by this switch include all those affected by the -Undefined_cell switch plus remaining port fault site definitions for existing cells and modules, as well as additional port fault sites without this scope.

- **-UNDEFined_cells**

  A required switch that automatically adds UDFM definitions for cell port fault sites in undefined (uncovered) cells or modules.

- -VERBose

  An optional switch that displays additional information about each added UDFM definition.

Example

The following example shows the effect of adding fault site definitions for all library cells not covered by UDFM definitions:

```
read_fault_sites /home/design/c090_std/c090_std.udfm
//  Reading UDFM File: /home/design/c090_std/c090_std.udfm
//     UDFM definitions read: Total=3, Cells=1, Modules=2, Instances=0, No Match=0
```

```
report_fault_sites -undefined_cells
//  Report of cells not defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  M1               2
//  M2               1
//  u_mux2           1
//  ---------  ---------

add_fault_sites -undefined_cells
//  3 UDFM definition(s) for uncovered cells have been added
add faults -all
//  Number added faults=56  instances considered=7  not-considered=3
```

Related Commands

add_faults
delete_faults
read_fault_sites
report_fault_sites
set_fault_type
write_fault_sites

add_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

add_faults ***fault_model_specific_options*** [*power_aware_options*]

Path Delay Faults Usage:
add_faults {***name_of_delay_path***… | **-All**} [-Both | -Rise | -Fall] [-VERbose]
[{> | >>} *file_pathname*]

Stuck/Transition/Toggle/Iddq Faults Usage:
add_faults {{ **-All** | {***object_expression***…
[-PIN | -INstance | -MODule [-PIN *module_pin_pathname* ...]]}} |
{**-CLOCK_Domains** {**ALL** | ***clock_pathname…***} [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains] } |
{**-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}}}
[-Stuck_at {01 | 0 | 1}] [-VERbose] [{> | >>} *file_pathname*]

Bridge Faults Usage:
add_faults {{**-All** | {***bridge_name***… -NAME} | {***net_pathname***… -SINGle}|
{***instance_name***… -SINGle}} |
{**-CLOCK_Domains** {**ALL** | ***clock_pathname…***} [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]} |
{**-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}} [{> | >>} *file_pathname*]

UDFM Faults Usage:
add_faults { **-All** | { object_expression... [-INstance] } }
| {**-CLOCK_Domains** {**ALL** | ***clock_pathname…***} [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains] }
{ [-Udfm_type *name*][-CEll *name*] [-MODule *name*] [-FAult *name*] } [-VERbose]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):
[[-ON_domains] | [-OFf_domains] | [-POWer_domains {*domain_name* …}]]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

Adds faults to the current fault list, discards all patterns in the current test pattern set, and sets all faults to undetected (actual category is UC).

When you enter the Setup mode, all faults from the current fault list are deleted. Furthermore, if you change the fault type, all faults are deleted. The power-aware options add faults based on power domains and/or power features after you have loaded a CPF or UPF file.

You cannot add faults for clock domains, capture procedures, and specific objects or paths within the same command instance. Use a separate command instance for each.

With the fault mode set to “uncollapsed” (the invocation default), all possible faults are added to the list. If you change the fault mode to “collapsed” with the set_fault_mode command, only one instance of any given fault is added, ignoring any equivalent faults.

Note

The add_faults command does not add faults that are protected as a result of a prior “read_faults -protected” or “set_fault_protection on” command to the internal fault list. Nor does it add faults that are equivalent to the protected faults; these faults are added to the internal list of protected faults.

Arguments

- **-All**

  A required switch that adds all faults, depending on the fault model.

  - **Bridge Faults** — Adds all previously loaded bridge entries to the internal fault list. This is the default behavior.
  - **UDFM Faults** — Adds faults for all previously loaded UDFM fault sites to the internal fault list. This is the default behavior.
  - **Non-Bridge Faults** — Adds all faults on all model, netlist primitive, and top module pins. The setting of the set_internal_fault command affects the behavior of this switch. If the set_internal_fault command is set to “off” (the default), the switch only adds faults residing on the boundaries of library cells. If the set_internal_fault command is set to “on”, the -All switch adds faults that reside on the inputs and outputs of gates within library cells. For more information about fault locations, refer to “Fault Locations” in the *Tessent Scan and ATPG User’s Manual*.

- ***name_of_delay_path***

  A required, repeatable string that specifies the names of the delay paths whose faults you want added to the current fault list. The name must correspond to the name of a delay path defined in the path definition file.

- -Both | -Rise | -Fall

  An optional switch that specifies which faults to add for each path already added via the Add Paths command. These switches are used for path delay faults only.

  -Both — Adds both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise — Adds only the slow-to-rise faults.

  -Fall — Adds only the slow-to-fall faults.

- ***object_expression***

  A required, repeatable string that specifies a list of instances or pins whose faults you want added to the current fault list. Note that in the case of UDFM, you can specify only instances. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

Instance pathnames must be ATPG library cell instances. Pin pathnames must be ATPG library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of an ATPG library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -PIN

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then add faults for all the pins matched.

- -INstance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then add faults for all the pins on the instances matched.

- -MODule [-PIN *module_pin_pathname* ...] (Stuck/Transition/Toggle/Iddq only)

  A switch that specifies to interpret the object_expression argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the object_expression argument, and the tool adds the fault for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and adds faults for every pin specified in module_pin_pathname list.

- ***bridge_name***

  A required, repeatable string that specifies bridge names whose bridge faults you want added to the current fault list.

- ***net_pathname***

  A required, repeatable string that specifies single net names whose bridge faults you want added to the current fault list.

- ***instance_name***

  A required, repeatable string that specifies instances names whose bridge faults you want added to the current fault list.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

  If the string specifies a net, faults are added to the fault list for each bridge entry associated with the net.

If the string specifies an instance that contains a non-dominant net specified in a bridge entry, associated faults are added to the fault list.

- **-CLOCK_Domains** {**ALL** | ***clock_pathname…***}

  A required switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to add only faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock.

  When you use this switch, the tool ignores the constraint values implied in the data path when adding faults in the data path. However, the tool still considers the constraint value for the clock cone tracing to determine the state element clock domain.

  The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

  clock_pathname — A repeatable string that specifies individual clocks.

Note

A fault added using this switch might also be detectable by an unspecified clock.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from adding faults in equivalent clock domains.

- -INCLUDE_INTER_NORACE_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to take user-defined non-race clocks into account when adding faults in the clock domain. You can use this switch in non-setup mode and when the clock restriction is set to be domain_clock.

- **-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}

  A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to add faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  For information about enabling or disabling named capture procedures, see the set_capture_procedures command description.

- -Stuck_at {01 | 0 | 1}

  An optional switch and literal pair that specifies which stuck-at or transition faults to add to the fault list.

01 — A literal specifying that, for stuck-at faults, the tool add both the “stuck-at-0” and “stuck-at-1” faults; or for transition faults, the tool add both “slow-to-rise” and “slow-to-fall” faults. This is the default.

0 — A literal specifying that, for stuck-at faults, the tool add only the “stuck-at-0” faults; or for transition faults, the tool add only “slow-to-rise” faults.

1 — A literal specifying that, for stuck-at faults, the tool add only the “stuck-at-1” faults; or for transition faults, the tool add only “slow-to-fall” faults.

- -Udfm_type *name*

  A switch and string pair that specifies a UDFM type. Use this switch to add faults associated with a specified UDFM type. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -CEll *name*

  A switch and string pair that specifies a library cell. Use this switch to add faults associated with a specified library cell. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -MODule *name* (UDFM only)

  A switch and string pair that specifies a module. Use this switch to add faults associated with a specified module. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -VERbose

  By default, when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, a summary message similar to the following is output:

  // Note: Adding faults for 330 fault sites.

  When you specify the optional -VERbose switch, the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

  When you specify the -Udfm switch and include the -Verbose option, the tool outputs a message similar to the following:

  // Note: Adding 627224 for 27398 instances

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that adds faults on all power-on domains.

- -OFF_domains

  An optional switch that adds faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that adds faults on the specified power domains (*domain_name*).

- -Isolation_cells

  An optional switch that adds faults on all isolation cells.

- -Level_shifters

  An optional switch that adds faults on all level shifters.

- -REtention_cells

  An optional switch that adds faults on all retention cells.

Example 1

The following example adds all faults to the circuit so you can run an ATPG process:

set_system_mode analysis
create_patterns

Example 2

The following example uses a prefix match to add faults only to pins on instances whose pathnames begin with “/u9/u2/LCT_reg”:

add_faults /u9/u2/LCT_reg*

```
=== Found 16 design pins ===
/u9/u2/LCT_reg_0_/A0
/u9/u2/LCT_reg_0_/A1
/u9/u2/LCT_reg_0_/S0
/u9/u2/LCT_reg_0_/Y
/u9/u2/LCT_reg_1_/A0
/u9/u2/LCT_reg_1_/A1
/u9/u2/LCT_reg_1_/S0
/u9/u2/LCT_reg_1_/Y
/u9/u2/LCT_reg_2_/A0
/u9/u2/LCT_reg_2_/A1
/u9/u2/LCT_reg_2_/S0
/u9/u2/LCT_reg_2_/Y
/u9/u2/LCT_reg_3_/A0
/u9/u2/LCT_reg_3_/A1
/u9/u2/LCT_reg_3_/S0
/u9/u2/LCT_reg_3_/Y
```

Example 3

The following example uses the same prefix match as the preceding example, but includes the -Instance switch:

delete_faults -all
add_faults /u9/u2/LCT_reg* -instance

```
=== Found 4 design instances ===
/u9/u2/LCT_reg_0_
/u9/u2/LCT_reg_1_
/u9/u2/LCT_reg_2_
/u9/u2/LCT_reg_3_
```

Notice that when you use the -Instance switch, the command lists the instances whose pins are faulted as a result of the wildcard character rather than listing each individual pin.

Example 4

The following example demonstrates how to wildcard the second level of hierarchy:

add_faults /u9/*/LCT_reg_0_ -instance

```
=== Found 4 design instances ===
/u9/u4/LCT_reg_0_
/u9/u3/LCT_reg_0_
/u9/u2/LCT_reg_0_
/u9/u1/LCT_reg_0_
```

Example 5

The following example combines a wildcard character at the second level of hierarchy with a prefix match:

add_faults /u9/*/LCT_reg* -instance

```
=== Found 32 design instances ===
/u9/u4/LCT_reg_0_
/u9/u4/LCT_reg_1_
/u9/u4/LCT_reg_2_
/u9/u4/LCT_reg_3_
/u9/u4/LCT_reg_4_
/u9/u4/LCT_reg_5_
/u9/u4/LCT_reg_6_
/u9/u4/LCT_reg_7_
/u9/u3/LCT_reg_0_
/u9/u3/LCT_reg_1_
/u9/u3/LCT_reg_2_
...
/u9/u2/LCT_reg_5_
/u9/u2/LCT_reg_6_
/u9/u2/LCT_reg_7_
/u9/u1/LCT_reg_0_
/u9/u1/LCT_reg_1_
/u9/u1/LCT_reg_2_
/u9/u1/LCT_reg_3_
/u9/u1/LCT_reg_4_
/u9/u1/LCT_reg_5_
/u9/u1/LCT_reg_6_
```

```
/u9/u1/LCT_reg_7_
```

Example 6

The following example adds only faults in the power domain named PD1:

```
add_faults -power_domain PD1
```

Example 7

The following example uses a prefix match to add faults to pins on instances whose pathnames begin with “/u9/u2/LCT_reg” plus all level shifters in the design:

```
add_faults /u9/u2/LCT_reg* -level_shifters
```

Example 8

The following example adds *intra_cell_bridges* UDFM fault models associated with all library cells that start with *mux* to the current fault list.

```
add_faults -all -udfm_type intra_cell_bridges -cell mux*
```

Example 9

The following example adds intra_cell_bridges UDFM fault models to the instance my_module1_ins/my_module1_or02. Note, when adding a UDFM fault model, the -instance switch is not used.

```
add_faults my_module1_ins/my_module1_or02 -udfm_type intra_cell_bridges
```

Example 9

The following example adds fault models to every module in the current fault list whose name begins with *adder*.

```
add_faults -all -module adder*
```

Example 10

The following example displays the fault value on the specified gate_pin after the add_faults command is executed:

```
add_nofaults ha1/ddd/q -Pin -Stuck 1
add_nofaults ha1/ddd/clk -Pin -Stuck 0
add_nofaults ha1/ddd/d -Pin -Stuck 01
....
add_faults -all
....

foreach_in_collection i [get_gate_pins ha1/ddd/* ] {
        puts "[get_name_list $i] fault_site: [get_attribute_value_list $i -name fault_site]"

}
```

```
ha1/ddd/clk   fault_site: 1
ha1/ddd/d   fault_site: none
ha1/ddd/q   fault_site: 0
```

Related Commands

add_norace_clocks
create_patterns
delete_faults
read_faults
read_cpf
read_upf
report_faults
report_testability_data
set_fault_mode
set_fault_protection
set_fault_sampling
set_fault_type
set_internal_fault
write_faults

add_icl_ports

Context: patterns -ijtag

Mode: setup

Usage

add_icl_ports ***port_objects*** [-type { data_in | data_out}]

Description

Specifies top design ports that will be added as DataInPort or DataOutPort ports in the ICL file generated during ICL extraction.

You must specify the current design with the set_current_design command before issuing this command. Changing the current design will discard all previously specified DataInPort or DataOutPort ICL ports. The tool validates that the ports specified as data_in are input or bidirectional ports, and ports specified as data_out are bidirectional or output ports.

Note

You must load the design (gate-level Verilog or RTL) before you issue the set_current_design command as the tool verifies that the specified ports exist in the portlist. You may also load the top level design as a black-box module.

During ICL extraction, top level ports in the controlling fanin of DataInPort ports of sub-modules will automatically be inferred as DataInPort ports and top level ports in the controlling fanout of DataOutPort ports of sub modules will automatically be inferred as DataOutPort ports. You use this command if you want the top level ICL file to declare DataInPort ports and/or DataOutPort ports even when they don't have any DataInPort or DataOutPort ports in their fanin or fanout. Once a port is identified as a DataInPort in the ICL file, it can then be set using the iWrite command. Once a port is identified as a DataOutPort in the ICL file, it can then be observed using the iRead command. Later on, when the current design is integrated into a higher module, ICL extraction will trace the newly created DataInPort ports and DataOutPort ports and ijtag retargeting can be used to retarget the iWrite and iRead commands to the boundary of the new top module.

Arguments

- ***port_objects***

  A required string that specifies a design port name, a tcl list of design port names or a collection of design ports. An error is generated if the specified ports are not found in the portlist of the design.

- -type { data_in | data_out }

  An optional switch and literal pair that specifies whether the specified ports are labeled as DataInPort ports or DataOutPort ports in the generated ICL module. If the -type option is not specified, the ICL port type will be inferred as DataInPort for input and bidirectional

ports and DataOutPort for output ports. An error is generated if -type data_in is specified for an output port or -type data_out is specified for an input port.

Example

The following example declares an ICL DataInPort port D1, and three ICL DataOutPort ports O1 O2 and O3. The generated ICL module will include “DataInPort D1;”, “DataOutPort O1;”, “DataOutPort O2;” and “DataOutPort O3;” even if no ICL ports traced to or from them in the design.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
```

Related Topics

delete_icl_ports

add_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Usage

add_icl_scan_interfaces ***name_list***

Description

Adds the specified scan interfaces to the new ICL top module that will be created during ICL extraction.

You only need to use this command if the top module on which ICL extract is running has more than one ScanInterface. In most cases, even when the top module has more than one ScanInterface, ICL extract is able to infer them correctly from the connections to and from the lower-level ICL modules. Only in complicated cases, the heuristic used by ICL extraction may fail; in this case, you use this command to describe the ScanInterface of the top module and they will be reflected in the generated ICL of the top module.

Every ScanInterface must have a port list, so after using the add_icl_scan_interfaces command, use the set_icl_scan_interface_ports command to set the list of ports for each ScanInterface.

You must set the current design with set_current_design before using the add_icl_scan_interfaces command.

Arguments

- ***name_list***

  A required list that specifies the names of scan interfaces that ICL extraction will create in the new ICL top module.

Example

In the following example, the add_icl_scan_interfaces command will create ScanInterfaces named TAP and Internal in the new ICL top module during ICL extraction. The set_icl_scan_interface_ports command will add the names of the ports to the TAP and Internal ScanInterfaces.

set_current_design tap1

add_icl_scan_interfaces {TAP Internal}

set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}

set_icl_scan_interface_ports -name Internal -ports {tdo tdrEn1 fromTdr1 ce se ue}

The resulting ICL model would look like this:

```
Module tap1 {

   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
```

```
    TMSPort         tms;
    TRSTPort        trst;
    ToSelectPort    tdrEn1    {  Source sel1;  }
    ScanInPort      fromTdr1;
    ToCaptureEnPort ce;
    ToShiftEnPort   se;
    ToUpdateEnPort  ue;

    ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}
    ScanInterface Internal {Port tdo; Port tdrEn1; Port fromTdr1; Port ce;
Port se; Port ue;}
```

Note

Every ScanInterface must have a port list. After using the add_icl_scan_interfaces command, use the set_icl_scan_interface_ports command to set the list of ports for each ScanInterface.

Related Commands

delete_icl_scan_interfaces
get_icl_scan_interface_list
get_icl_scan_interface_port_list
set_icl_scan_interface_ports

add_iddq_exceptions

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

Mode: setup, analysis

Usage

add_iddq_exceptions ***ZVAL_gate…***

Description

Enables you to specify individual sites where the tool does not apply the Iddq restrictions.

You can achieve maximum quiescence by using the appropriate IDDQ checks, and adding exceptions for as few sites as possible while still obtaining patterns that are not rejected for violation of IDDQ checks during fault simulation.

For more information on IDDQ, refer to "Creating an IDDQ Test Set" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***ZVAL_gate…***

  A required repeatable string that declares the gate name or gate identification number of a site for which the tool could report Iddq check violations. (See the set_iddq_checks command for specifics about setting Iddq checks). These sites are always roots of tri-state buses (ZVAL gates at the primitive level).

Note

If a site does not have a unique name, you must provide a gate identification number.

The "report_gates -type zval" command gives a list of all candidates, but typically you only need to use the gate identification numbers reported by the analyze_restrictions command.

Example

The following example adds Iddq restrictions, uses the analyze_restrictions command to locate the sources of violations, then adds an Iddq exception to remove one of them.

set_iddq_checks -int_float
simulate_patterns
analyze_restrictions

```
...
//  Will attempt to locate and report specific -int_float violators.
//  No single bus is constrained to Z (or already Z).  Will analyze
further.
//  The following ZVAL gates cause -int_float to fail by themselves.
//    ZVAL gate /cpu/alu/U2392/ (54417).  ABORTed.
//    ZVAL gate /cpu/alu/U4177/ (63712).  ABORTed.
...
```

add_iddq_exceptions /cpu/alu/U2392/

To use the analyze_restrictions command most effectively, you must have design knowledge; in some cases, design knowledge is required in order to pinpoint the source of the violation.

Related Commands

analyze_restrictions
report_gates
report_primary_outputs
set_iddq_checks

add_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Usage

add_ijtag_logical_connection **-from** *source_pin_or_port_name*
-to *destination_pin_or_port_name*

Description

Creates a new logical connection path through the current design by specifying the source and destination of the logical path.

During ICL connection tracing, logical connections will be followed in preference to the actual connections. When tracing reaches an instance pin with a logical connection, it will follow the logical connection across the design hierarchy. A forward trace will only follow logical connections that start on the pin being traced and then skip to the destination of the logical connection. A backward trace will only follow logical connections that have a destination on the pin being traced and then skip to the source of the logical connection.

See "Example 1" on page 479 for an example of a logical connection .

Use this command with care as it basically bypasses the structural tracing of the circuit and uses an act of faith on the behavior of the specified logical source. If the behavior turns out to be inconsistent with the specified logical sources, you will only detect the issue in simulation. Use this command to model logical sources through blackbox modules or to model a path where the logic is too complex. A circuit that gates TCK with capture and shift enable provides proper capture and shift behavior but would not be properly traceable by ICL extraction. In such a case, using a logical source to allow tracing through the tck gating logic is needed.

The following attributes are used to store the source...destination relationship for the two ends of the logical connection:

Table 7-6. Logical Connection Attributes

Attribute Name	Usage
ijtag_logical_hier_connection_from_src	Associated with the top-level module port or instance pin that is the destination of an hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the source ends of of these logical connections. This attribute is not inherited but does cause the the hierarchical boundary to be preserved.
ijtag_logical_hier_connection_to_dst	Associated with the top-level module port or instance pin that is the source of an hierarchical logical connection. A string list of top level module port names and hierarchical instance pin path names to the destination ends of of these logical connections. This attribute is not inherited but does cause the the hierarchical boundary to be preserved.

The following figure illustrates the usage of these attributes.

Figure 7-5. Logical Connection Attributes

These attributes are available for direct read only introspection by the user, like the other predefined ICL extraction attributes. They will all be cleared when the set_current_design command is executed.

Arguments

- **-from** *source_pin_or_port_name*

  A required switch and value pair that specifies the source point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to** *destination_pin_or_port_name*

  A required switch and value pair that specifies the destination point of the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

Example 1

The following example illustrates a logical connection from pin U1/Y to pin U2/A. A forward trace that arrives at pin U1/Y will continue on from pin U2/A. A backward trace that arrives at pin U2/A will continue on from pin U1/Y.

Figure 7-6. Logical Connection Example

At the netlist level all interconnections start at an instance pin or a top level module port and end at an instance pin or top level module port. Establishing a logical connection within the design hierarchy will not alter the way that that design hierarchy is traversed by ICL extraction tracing when that design hierarchy is used as part of a larger design hierarchy, from a different top level module.

Example 2

The add_ijtag_logical_connections command will create logical connection between the pins /i2/in and /i2/out, /i3/in and /i3/out and i4/in and /i4/out. The report_ijtag_logical_connections command will report all logical connections in the current design.

add_ijtag_logical_connection -from /i2/in -to /i2/out

add_ijtag_logical_connection -from /i3/in -to /i3/out

add_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination
//  ===========  ==============
//  /i2/in       /i2/out
//  /i3/in       /i3/out
//  /i4/in       /i4/out
//  /i5/in       /i5/out
```

Related Commands

delete_ijtag_logical_connection

report_ijtag_logical_connections

add_input_constraints

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Usage

Context: dft -scan and dft-test_points

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[***constraint_switch***]}}

Context: patterns -scan

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[***constraint_switch***]}}
[-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]]

Context: patterns -ijtag

add_input_constraints
{{[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude]
constraint_switch} |
{[-EQuivalent *reference_pin* [*equivalent_pin*]… [-INvert *inverted_pin*…]]
[***constraint_switch***]}}

Description

Constrains primary inputs to specified values.

The add_input_constraints command constrains primary input pins to specified values.

Note

Do not force a constrained pin to a new value in any cycle of a named capture procedure (NCP), even temporarily. To ensure state stability, a constrained pin's state must be kept constant for all cycles of NCPs.

Note

In the patterns -ijtag context, you can constrain all Verilog ports unless the port is also an ICL port. Input constraints on ICL ports are allowed only for ICL ports of type TRSTPort and ClockPort. For more information, see "Constraining Inputs" in the Tessent IJTAG User's Manual.

Arguments

- -ALl

  An optional switch that applies the constraint to all applicable primary input pins except clocks, read controls, and write controls. If -Slow_pad is also specified, the constraint is only applied to primary input pins that can take a Z value. This is the default.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin to which to apply the constraint.

- -BIDI_Only

  An optional switch that specifies to apply the constraint to bidirectional pins only.

- -BIDI_Exclude

  An optional switch that specifies to apply the constraint only to primary input pins that are not bidirectional.

- -SCAN_INputs

  An optional switch that specifies to apply the constraint to scan input pins only.

- -EQuivalent *reference_pin* [*equivalent_pin*]…

  An optional switch, string, and repeatable string triplet that specifies for the *primary_input_pin_name*(s) to take on the value of the *reference_pin*. If you optionally specify one or more of the *primary_input_pin_name*(s) as *equivalent_pin*(s), those PI's in the *equivalent_pin* list take on the value of the *reference_pin*.You must list pins you want to be equivalent in value to the reference pin before the -Invert argument when used.

- -INvert *inverted_pin*…

  An optional switch and repeatable string that together with the -Equivalent switch specify for the primary input pins, *inverted_pin*, to take on the inverted value of the *reference_pin*. You must list the pins you want to be inverted in value after the -Invert argument.

- ***constraint_switch*** or *constraint_switch*

  A required or optional switch (depending on the arguments you use with it) that specifies the constant value to which to constrain the primary input pin(s). The choices, from which you can specify only one, are as follows:

  -C0 — Specifies application of the constant 0 to the chosen primary input pin.

  The tools do not use constant 0 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use a -C0 constraint if the constraint is valid in test mode, but *not* valid in system mode.

Note

If the constraint is valid in system mode as well, consider using -CT0 instead. Faults associated with tied circuitry resulting from a -CT0 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C0 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT0 only for valid system mode constraints.

-C1 — Specifies application of the constant 1 to the chosen primary input pin.

The tools do not use constant 1 constraints to determine tied circuitry. Therefore, faults associated with line holds resulting from these constraints are classified as ATPG_untestable (AU), as opposed to Tied (TI) or Blocked (BL). Use -C1 if the constraint is valid in test mode, but *not* valid in system mode.

Note

If the constraint is valid in system mode as well, consider using -CT1 instead. Faults associated with tied circuitry resulting from a -CT1 constraint are not classified as ATPG_untestable, thereby improving test coverage compared to using a -C1 constraint. To ensure the improvement in coverage matches reality, however, be sure to use -CT1 only for valid system mode constraints.

-CX — Specifies application of the constant X (unknown) to the chosen primary input pin. The tools do not use the specified primary input pin for control.

-CZ — Specifies application of the constant Z (high-impedance) to the chosen primary input pin. The CZ constraint is only applicable to bidi pin and PI that fans out only to Z consumer gates, such as PO, or bus. For a PI that fans out to non Z consumer gate, when CZ is added the tool will issue a warning message during the flattening process and convert CZ constraint into CX constraint automatically. For example:

```
// Warning: Invalid constant Z pin constraint on pin te5 changed to
// constant X.
```

-CT0 — Specifies application of the constant TIE0. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT0 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are *not* included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C0 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT0 instead of a -C0 constraint; but take care that any -CT0 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

You cannot override a -CT0 value (from a test procedure file for example). If you need this flexibility, use -C0 instead.

-CT1 — Specifies application of the constant TIE1. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CT1 pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -C1 pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CT1 instead of a -C1 constraint; but take care that any -CT1 constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

Note

You cannot override a -CT1 value (from a test procedure file for example). If you need this flexibility, use -C1 instead.

-CTZ — Specifies application of the constant TIEZ. The ATPG process treats this constraint as if you had added a tied signal.

The tools classify faults associated with tied circuitry resulting from a -CTZ pin constraint as untestable (Tied (TI) or Blocked (BL)). These faults are not included in the calculation of test coverage, so they neither raise nor lower coverage. If instead, the faults were associated with a -CZ pin constraint, the tool would classify them as ATPG_untestable (AU) and they would reduce coverage. Therefore, it is advantageous to use a -CTZ instead of a -CZ constraint; but take care that any -CTZ constraint you apply for test is valid in system mode—to ensure the relative boost in coverage is real.

- -HOld

  An optional switch that, for clock sequential patterns only, holds the constant constraint value at the primary input(s) for all cycles of a clock_sequential or multiple_load pattern. Even if a pattern has multiple loads, the primary input's value cannot change among different load intervals. The hold does not apply to clocks, read controls and write controls. It also does not apply to launch off shift patterns, RAM sequential patterns, or patterns with a clock procedure.

  If the fault type is set to transition and you enter a "set_transition_holdpi On" command, the constraint defined by the add_input_constraints command is overridden by the set_transition_holdpi command; that is, all primary inputs except clocks and read/write controls are going to hold their values even if you have not applied a constraint to a particular primary input. Similar behavior occurs when the fault type is set to path delay and you enter a "set_pathdelay_holdpi On" command.

- -NO_Z

  An optional switch that specifies not to allow a Z value at the specified primary input pin(s) during test generation, even if the pins can support a Z value. During good machine simulation, a pattern is rejected if it assigns a Z value to any primary input that is specified to have no Z value. This switch is invalid with either the -CZ or -CTZ switch.

- -SLow_pad

  An optional switch that specifies to X out the value at a bidirectional pin if the pin is driven from within during test generation.

  -CELL *model_name*

  An optional switch and string pair that together with the -Slow_pad switch specifies the scope of the bidirectional pins to be included as slow pads. Only the bidirectional pins that drive the instances initiated from the specified library cell name are added as slow pads.

Example 1

The following example shows two differential clocks, CLK1 off state 0 and CLK2 off state 1. Since the pins are equivalent, you only need to add the clock to one of the pins.

SETUP> **add_input_constraints -equivalent CLK1 -invert CLK2**

SETUP> **add_clocks 0 CLK1**
// Note: Equivalent pin CLK2 also added to clock list.

Example 2

The following example shows how to add equivalent clocks that are inverted.

SETUP> **add_clocks 0 mode[1] -free_running**

SETUP> **add_clocks 1 mode[2] -free_running**

SETUP> **report_clocks**

```
User-defined Clocks (2):
---------  ------------  --------  -------  ----------------
Off State   Constraints  Internal  Name     Other Properties
---------  ------------  --------  -------  ----------------
     1      Free running     No     mode[2]
     0      Free running     No     mode[1]
```

SETUP> **add_input_constraints -equivalent mode[1] -inv mode[2]**

SETUP> **report_input_constraints**

```
//  Primary Input  Equivalent  C-Type  Hold      No-Z      Slow-Pad
//  -------------  ----------  ------  ----  ------------  --------
//  mode[1]                     NONE    No   Undetermined     No
//                 ~mode[2]     NONE    No   Undetermined     No
```

Related Commands

delete_input_constraints

report_input_constraints

add_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

add_lfsr_connections ***pin_pathname lfsr_name position***…

Description

Connects an external pin or an internal pin to a Linear Feedback Shift Register (LFSR).

The add_lfsr_connections command connects a core logic pin to an LFSR. You specify this pin with the *pin_pathname* argument. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes that the output of the 0 bit position connects to the input of the highest bit position. If you select multiple bits of a Pseudo-Random Pattern Generator (PRPG) for the *position* argument, the tool assumes they are all exclusive-ORed together to create the value for the pin.

If you determine that multiple pin names must connect to a bit position of a Multiple Input Signature Register (MISR), you must issue a separate add_lfsr_connections command for each pin. The tool assumes the pins are all exclusive-ORed together to create the value for the next MISR input. The tool also assumes that the physical placement of the MISR connections is after the tapping points as shown in Figure 7-7.

Figure 7-7. MISR placement

You can use the report_lfsrs command to display all the LFSRs with their current values and tap positions.

Arguments

- ***pin_pathname***

  A required string that specifies the pin pathname (*<instance pathname>/<pinname>*) of the core logic pin that you want to connect to the LFSR specified by *lfsr_name*.

- ***lfsr_name***

  A required string that specifies the name of the LFSR to which you want to connect the *pin_pathname*.

- ***position***

  A required repeatable integer that specifies the bit positions of the *lfsr_name* at whose outputs you want to place connections. A bit position is an integer number, where 0 indicates the least significant bit position. The tool assumes the output of the 0 bit position connects to the input of the highest bit position.

Example

The following example connects an LFSR to a scan-in pin and another LFSR to a scan-out pin:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
add_lfsr_connections scan_in.1 lfsr1 2
add_lfsr_connections scan_out.0 misr1 3
```

Related Commands

add_lfsrs
add_lfsr_taps
delete_lfsr_connections
report_lfsr_connections

add_lfsr_taps

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

add_lfsr_taps ***lfsr_name position***…

Description

Adds the tap configuration to a Linear Feedback Shift Register (LFSR) or a PRPG of type cellular automata (CA).

The add_lfsr_taps command sets the tap configuration of an LFSR. LFSR bit positions have integer numbers, where 0 indicates the least significant bit position. The tool assumes the output of the 0-bit position connects to the selected tap points and that the 0-bit position cannot itself be a tap point.

If the LFSR is defined as a CA, the tap positions define the position of the self-feedbacks cells (those CA cells defined using rule 150). At least one tap must be defined; otherwise, the consistency check of the PRPG issues an error message.

You can use the report_lfsrs command to display all of the LFSRs with their current values and tap positions. You can change the default setting of the *tap_type* switches by using the set_lfsrs command.

Arguments

- ***lfsr_name***

  A required string that specifies the name of the LFSR on which you want to place the taps.

- ***position***

  A required, repeatable integer that specifies the bit positions of the *lfsr_name* at whose outputs you want to place the taps or, for CAs, the bit positions of the self-feedback cells.

Example 1

The following example places taps on the newly-added LFSRs:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```

Example 2

The following example defines a 32-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, 23, 25, and 30:

```
add_lfsrs my_prpg prpg 32 FFFFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23 25 30
```

Related Commands

add_lfsrs
delete_lfsr_taps
report_lfsrs
set_lfsrs

add_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

add_lfsrs ***lfsr_name*** {**Prpg** | **Misr**} ***length seed*** [-Both | -Serial | -Parallel]
[-Out | -In | -Ca [-INCrement_step *value*]]

Description

Adds LFSRs for use as PRPGs or MISRs.

The add_lfsrs command defines LFSRs, which the tool uses as PRPGs, to create pseudorandom values for the BIST patterns or as MISRs to compact responses.

You specify the LFSR's shift technique by using one of the following shift_type switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following tap_type switches: -Out or -In; or you specify the type of LFSR as celluar automata using the *tap_type* switch -*Ca*. You can change the default setting of the shift_type and tap_type switches by using the set_lfsrs command.

Arguments

- ***lfsr_name***

  A required string that specifies the name that you want to assign to the LFSR. Note that you cannot use the keyword "all" for this name.

- **Prpg**

  A literal that indicates the LFSR functions as a PRPG.

- **Misr**

  A literal that indicates the LFSR functions as a MISR.

- ***length***

  A required integer, greater than 1, that specifies the number of bits in the LFSR.

- ***seed***

  A required, right-justified, hexadecimal number, greater than 0, that specifies the initial state of the LFSR.

The following lists the shift_type switches of which you can choose only one.

- -Both

  An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default unless you change it with the set_lfsrs command.

- -Serial

  An optional switch specifying that the LFSR shifts serially the number of times equal to the length of the longest scan chain for each scan pattern.

- -Parallel

  An optional switch specifying that the LFSR parallel shifts once for each scan pattern.

The following lists the three tap_type switches of which you can only choose one.

- -Out

  An optional switch specifying that the exclusive-OR taps reside outside the register path. This is the default unless you change it with the set_lfsrs command.

- -In

  An optional switch specifying that the exclusive-OR taps reside in the register path.

- -CA

  An optional switch specifying that the PRPG is of type *cellular automata*. This switch is only valid if the LFSR type is specified as *Prpg*.

- -INCrement_step *value*

  An optional switch that specifies the number of times the PRPG is advanced for each shift cycle. The *value* argument is a required integer.

Example 1

The following example defines an LFSR to be a PRPG and another LFSR to be a MISR:

```
add_lfsrs lfsr1 prpg 5 10 -serial -in
add_lfsrs misr1 misr 5 15 -both -out
add_lfsr_taps lfsr1 1 3
add_lfsr_taps misr1 1 2
```

Example 2

The following example defines an LFSR to be a PRPG without a phaseshifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Example 3

The following example defines an LFSR to be a PRPG with a phaseshifter:

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsr_connections scanin_1 my_prpg 0 3 5
add_lfsr_connections scanin_2 my_prpg 1 7 11
...
add_lfsr_connections scanin_24 my_prpg 9 15 23
```

Example 4

The following example defines an LFSR to be a PRPG and specifies to advance the PRPG eight times for each shift cycle.

```
add_lfsrs my_prpg prpg 24 FFFFFF -CA -inc 8
add_lfsr_taps my_prpg 3 7 12 15 23
add_lfsrs connection scanin_1 my_prpg 0
add_lfsrs connection scanin_2 my_prpg 1
...
add_lfsrs connection scanin_24 my_prpg 23
```

Related Commands

add_lfsr_taps
add_lfsr_connections
delete_lfsrs
report_lfsrs
set_lfsrs
set_lfsr_seed

add_lists

Context: dft -edt, patterns -scan

Mode: analysis

Usage

add_lists ***pin_pathname***...

Description

Adds pins to the list of pins on which to report.

In “dft -edt” context, you cannot report values on gates in the EDT logic with this command because values inside the EDT logic are not always accurate.

Tip: This command is a useful debugging aid, as it provides a way to check the values simulated for specific output pins of cells within the design.

To list all the pins on which the tool will currently report during simulation, use the report_lists command. To turn off reporting of a pin’s simulation values, use the delete_lists command. To direct the tool to write the pins’ simulation values to a file instead of displaying them onscreen, use the set_list_file command.

When switching to Setup mode, the tool discards all pins from the report list.

Arguments

- ***pin_pathname***

  A required, repeatable string that specifies the output pins on which to report during good and/or faulty machine simulation.

Example

The following example writes to the file, *pinlist_goodsim.txt*, the values simulated for two output pins during simulation of an external pattern source:

```
set_system_mode analysis
read_patterns pattern1
add_lists /i_1006/o /i_102/o
set_list_file pinlist_goodsim.txt -replace
simulate_patterns
```

The following shows the format of the resulting list file: the first column lists the pattern index of each pattern simulated, the second column shows the value simulated on each monitored pin for each pattern. Note that the pattern index refers to the simulated parallel patterns rather than the effective patterns that will be stored after ATPG.

```
//        /i_1006/o
//        |/i_102/o
//        ||
//        ||
//----------
```

```
//      0  11
//      1  10
//      2  00
//      3  10
//      4  01
//      5  01
...
```

Notice that in analysis system mode, the report shows the good value of the pin. The tools also shows the values at each test pattern.

The following is example list file content for the same two pins when simulated in the analysis system mode:

```
//        /i_1006/o
//        |/i_102/o
//        ||
//        ||
//----------
//      0  11
//      0  0- { /i_1006/o@0 }
//      1  10
//      1  0- { /i_1006/o@0 }
//      2  00
//      2  1- { /i_1006/o@1 }
//      3  10
//      3  0- { /i_1006/o@0 }
//      4  01
//      4  1- { /i_1006/o@1 }
//      5  01
//      5  1- { /i_1006/o@1 }
...
```

The report now includes two lines of information for each pattern simulated. The first line shows the good machine value of each pin as before. The second line shows the faulty value of each pin if different from the good value (a “-” indicates the faulty value was the same as the good value). Inside the “{}” is the fault site and its faulty machine stuck-at value the tool simulated to produce the faulty machine results.

Related Commands

delete_lists
report_lists
set_list_file

add_misrs

Context: patterns -scan

Mode: setup, analysis

Usage

add_misrs ***misr_name*** **-size** ***int*** [-seed ***hex***] [-taps ***index***…]

Description

Specifies the MISR structure and initial seed.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

This command is included in the LogicBIST fault simulation dofile to describe the MISR logic synthesized.

Arguments

- ***misr_name***

  A required string that specifies the name of the MISR. The ***misr_name*** is subsequently used by the tool for referring to this MISR for specifying connections from scan chains, and in reporting.

- **-size** ***integer***

  A required switch and integer that specifies size of the MISR.

- -seed ***hexadecimal***

  An optional switch and hexadecimal number pair that specifies the initial seed. Use this command if you intend on performing fault simulation with a different starting MISR seed.

- -taps ***index***

  An optional switch and repeatable integers that specifies the tap points for the MISR feedback network. The tap index starts from 1 to be consistent with the decompressor used as a PRPG in the LogicBIST mode and associated decompressor commands.

Example

This example shows the add_misrs command in context patterns -scan, mode setup. This example adds a 24-bit MISR named "misr" with initial seed of all 0-s:

```
add_misrs misr -size 24 -seed 000000 -taps 21 22 24
```

Related Commands

delete_misrs

report_misrs

report_misr_connections

set_misr_connections

add_nofaults

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

Mode: setup

Usage

add_nofaults {{ ***modulename*** ... **-Module** [-PIN *module_pin_pathname* ...] | ***object_expression*** ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}] [-Keep_boundary]} | -Wire [-VERBose] [(> | >>) *file_pathname*]

Description

Places nofault settings either on pin pathnames, pin names of specified instances, or modules.

By specifying pathnames of pins, instances, or modules while in Setup mode, the add_nofaults command places a nofault setting either on the specific pins or on boundary and internal pins of the instances or modules. All added nofault pin pathnames are in the user class.

Note

All switches except -Wire are valid only before flattening. The -Wire switch is valid only after flattening.

The -Stuck_at switch applies to either stuck-at faults or transition faults. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a value, then the tool places a nofault setting on both stuck-at (transition) values. If you add faults with the add_faults command after you issue the add_nofaults command, the specified pin pathnames or boundary and internal pins of instances or modules cannot be sites for those added faults.

Note

When you add nofault settings, the tool deletes the flattened simulation model if it exists. This removes information you added after model flattening, such as ATPG functions. To avoid losing this information or having to wait for the tool to flatten the design again, add nofault settings prior to model flattening.

Arguments

- ***modulename***

  A repeatable string that specifies the name of a module to which you want to assign nofault settings. You must include the -Module switch when you specify a module name.

- **-Module** [-PIN *module_pin_pathname* ...]

  A switch that specifies to interpret the ***modulename*** argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***modulename*** argument, and the tool adds the nofault for all matching modules or library models.

When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given module name and adds nofaults for every pin specified in module_pin_pathname list.

- ***object_expression***

  A string representing a list of pathnames of instances or pins for which you want to assign nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

  Pin pathnames must be ATPG library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of an ATPG library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -PIN

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then assign nofault settings to all the pins matched.

- -Instance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then assign nofault settings to all boundary and internal pins of the instances matched (unless you use the -Keep_boundary switch).

- -Stuck_at <u>01</u> | 0 | 1

  An optional switch and literal pair that specifies to which stuck-at or transition values you want to assign a nofault setting. The choices are as follows:

  <u>01</u> — A literal that specifies to place a nofault setting on both the “stuck-at-0” and “stuck-at-1” faults for stuck-at faults; or on both the “slow-to-rise” and “slow-to-fall” faults for transition faults. This is the default.

  0 — A literal that specifies to place a nofault setting on only the “stuck-at-0” faults for stuck-at faults; or on only the “slow-to-rise” faults for transition faults.

  1 — A literal that specifies to place a nofault setting on only the “stuck-at-1” faults for stuck-at faults; or on only the “slow-to-fall” faults for transition faults.

- -Keep_boundary

  An optional switch that specifies to apply nofault settings to the inside of the specified instance/module, but allow faults at the boundary pins of these instances/modules. This option does not apply to nofaults on pin pathnames.

- -Wire

  A switch that specifies for the tool to no-fault a cone of logic fanning out from a stem forward into a single wire gate. The tool no-faults the entire cone of logic between the stem

and the wire gate, but not the stem or the output of the wire gate. There must be no other fanout from this cone of logic. This switch is valid only after flattening.

Note

This switch affects all qualifying logic cones.

- -VERbose

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

  // Note: Adding nofaults for 330 fault sites.

  When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example defines nofault settings for all the pins in a particular instance, so when you add all faults to the circuit for an ATPG run, the tool will not place faults on the pins of that instance:

```
add_nofaults i_1006 -instance
set_system_mode analysis
create_patterns
```

Example 2

The following example places nofault settings on all the design pins within all instances of wired cone logic, adds all faults to the circuit, and performs an ATPG run so that Tessent FastScan and Tessent TestKompress place nofaults on the wired cone logic pins:

```
set_system_mode analysis
add_nofaults -wire
create_patterns
```

Related Commands

add_faults
delete_faults
delete_nofaults
report_faults
report_nofaults

add_nonscan_instances

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

add_nonscan_instances { ***pathname...*** **|** ***instance_expression*** [-INStance | -Control_signal | -Module]} | -All

Description

Marks specific sequential elements for exclusion from the scan insertion process.

The add_nonscan_instances command causes the tool to ignore the specified instances, instances controlled by the specified control signals, all instances within the specified module, or all previously-identified scan instances when you execute the scan identification process with the insert_test_logic command.

Also, the tool will not perform the scannability rule checks on non-scan instances.

If you want to flag all instances of a certain library model as non-scan, you can use the add_nonscan_models command as a shortcut.

When adding non-scan instances with control signals, the tool adds all scannable instances controlled by the control signals to the non-scan instance list. Control pins can be any clock, set, or reset signal (primary input) that controls the contents of sequential instances. You can use the report_control_signals command to see the pins that control the sequential instances in the design.

If TIE0 and TIE1 non-scan cells are scannable, they are considered for scan. However, if these cells are used to hold off sets and resets of other cells so that another cell can be scannable, you must use this command to make them non-scan.

Arguments

- ***pathname***

  A required repeatable string that specifies the pathnames of the sequential instance or control signals (that control instances) which you want the tool to ignore. If the instance is hierarchical, then all instances beneath it are also flagged as non-scan.

- ***instance_expression***

  A string representing a list of instances within the design. The string instance_expression is defined as:

  { string | string * } ...

  The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. You can use a pathname expression to select several instances and the tool

will then add_nonscan_instances for all the pins on those instances; but if the expression specifies a location below the instance level of an ATPG library model, the tool will issue an error message.

- -INStance | -Control_signal | -Module

  An optional switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, …)” where clkgen is the module name. You can only use the -Control_signal option in analysis mode. The default is -Instance.

- -All

  An optional switch that marks every scan cell in the design as a non-scan cell. When you issue the command with the -All switch, the tool completely clears the list of targeted cells, and you can then manually select an arbitrary subset of target cells using the add_scan_instances command.

Example 1

The following example specifies that the tool ignore the sequential i_1006 instance when identifying and inserting the required scan circuitry:

```
add_nonscan_instances i_1006
```

Example 2

The following example clears all previously identified scan cells and then adds two instances to the list of scan cells to be stitched:

```
add_nonscan_instances -all
add_scan_instances i_1006 i_1007
insert_test_logic
```

Related Commands

add_scan_instances
delete_nonscan_instances
insert_test_logic
report_seq_transparent_procedures

add_nonscan_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

add_nonscan_models ***model_name***…

Description

Instructs the tool to ignore all instances of the specified sequential DFT library model when identifying and inserting the required scan elements and test logic into the design.

The add_nonscan_models command causes the tool to ignore the instances of the specified models, when you execute the scan identification process and scan insertion process with the insert_test_logic command.

The tool does not check the scannability on the instances instantiated from the added non-scan models, unless you specify checking using the set_lockup_cell command.

If you want to flag individual instances as non-scan, you can use the add_nonscan_instances command.

Arguments

- ***model_name***

  A required repeatable string that specifies the model names that you want the tool to ignore. You must enter the model names as they appear in the DFT library.

Examples

The following example specifies that the tool should not substitute any of the instances of the DFT library model named d_flip_flop with the equivalent scan replacement DFT library model.

```
add_nonscan_models d_flip_flop
```

Related Commands

delete_nonscan_models
insert_test_logic
report_nonscan_models
set_lockup_cell

add_norace_clocks

Context: dft -edt, patterns -scan

Mode: analysis

Usage

add_norace_clocks **-All** | ***clock_pin_name…*** | ***gate_id#…***

Description

Defines one or more non-interacting clocks.

The add_norace_clocks command defines the specified clocks as non-interacting. This gives you a way to override the tool's conservative algorithm for determining non-interacting clocks when you know with certainty that some clocks in your design are non-interacting. By specifying non-interacting clocks explicitly, you can often reduce generated test set size and improve test coverage.

Note

Simulation mismatch errors are likely if you erroneously define as non-interacting, clocks that actually do interact.

The tool includes all the clocks you list in one add_norace_clocks command in the same clock domain. A clock cannot be included in more than one user-defined clock domain, however, so once you define a clock as non-interacting, you cannot add another definition for it until you delete the existing definition using the delete_norace_clocks command. All clocks in the same clock domain may be turned on simultaneously by ATPG.

Arguments

- **-All**

  A switch that specifies all defined clocks.

- ***clock_pin_name…***

  A repeatable string that specifies the name of a defined clock pin.

- ***gate_id#…***

  A repeatable positive integer that specifies the gate identification number of a defined clock pin.

Example

The following example creates a user-defined clock domain for clocks "clk1" and "clk2". The example then tries to include clk1 in another user-defined clock domain. A clock can only belong to one user-defined clock domain, however, so the tool issues an error message. The example follows that with a new, valid definition, then displays the current user-defined clock

domains. Notice the user-defined clock domain does not alter the clock domains the tool determined from its own analysis. The latter are displayed by “report_clock_domains”.

set_system_mode analysis
add_norace_clocks clk1 clk2
add_norace_clocks clk1 clk5

```
// Error: Clock clk1 is already in a user-defined clock domain
```

add_norace_clocks clk3 clk4 clk5
report_norace_clocks

```
// User clock domain 1 : /clk1 /clk2
// User clock domain 2 : /clk3 /clk4 /clk5
```

report_clock_domains

```
// Clock name   Domain
// --------------------
// /clk1 (2)         1
// /clk2 (10)        2
// /clk3 (18)        3
// /clk4 (44)        4
// /clk5 (51)        5
```

Related Commands

delete_norace_clocks
report_clock_domains
report_norace_clocks
set_clock_restriction

add_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

add_notest_points {***pin_pathname***… | ***instance_pathname***… | ***instance_expression***} | **-Path** ***filename***

Description

Adds circuit points to the list for exclusion from testability insertion.

The add_notest_points command excludes the specified cell output pins, excludes specified D input scan cells, excludes all output pins on the specified instance, or excludes delay paths from use as controllability and observability insertion points. If the selected pin is already a control or observe point, an error occurs when you issue this command.

If you specify a path file that contains delay paths, the tool marks each gate in the path; consequently, the tool adds no test points to the output of that gate. This prevents control and observation points from being added to a critical path and thus prevents increasing the load on any of these gates in that path. The format of the file is the same as the file loaded with the read_fault_sites command. For more information on the format, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

The report_notest_points command displays all the pins in this list or can list the defined critical paths.

Arguments

- ***pin_pathname***

  A required, repeatable string that lists the output pins that you do not want to use for insertion of controllability and observability.

- ***instance_pathname***

  A repeatable string that lists the instances whose output pins you do not want to use for controllability and observability insertion. All output pins within that (hierarchical) instance are added to the list of pins that should be excluded from consideration.

- ***instance_expression***

  A string representing a list of instances within the design. The string *instance_expression* is defined as:

  { string | string * } ...

  The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. You can use a pathname expression to select several instances and the tool

will then add_notest_points for all the pins on those instances; but if the expression specifies a location below the instance level of an ATPG library model, the tool will issue an error message.

- **-Path *filename***

  A switch and filename pair that specifies the pathname to a file that contains critical path information. For more information on the format of the file, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

Example

The following example specifies output pins that the tool cannot use for testability insertion:

```
add_notest_points i_1006/o i_1008/o i_1009/o
```

Related Commands

delete_notest_points
report_notest_points
set_test_point_insertion_options

add_observe_points

Context: dft -test_points

Mode: setup, analysis

Usage

add_observe_points **-location** ***pin/port_spec***
[-scan_cell_clock ***pin/port_spec***]
[-enable ***pin/port_spec***]

Description

Specifies user-defined observe points during test point insertion.

Arguments

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

  A required string that specifies the pathname to the observe point location where ***pin/port_spec*** can be the name of a pin/port, a Tcl list of pins/ports, or a collection of pin/port objects.

- -scan_cell_clock ***pin/port_spec***

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

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

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

  If you want to specify the default enable signal for the automatically identified observe points instead of manually adding each enable signal, use the set_test_point_insertion_options command as follows:

  set_test_point_insertion_options -observe_points_enable

Example 1

The following example adds two observe points, and specifies the new scan cell model used to observe the inserted test points:

add_observe_points -location ss1/ix348/Z

add_observe_points -location ss1/ix302/Z

Example 2

The following example specifies the path names of existing nets that are to be used to enable/disable the control points and observe points respectively. Then it adds a control point and an observe point, and proceeds to insert the test points into the netlist:

```
set_test_point_insertion_options -control_point_enable cp_enable \
        -observe_point_enable op_enable

add_control_points -location jtc/i1216/OUT -type and

add_observe_points -location jtc/i1645/OUT

...

set_system_mode analysis

...

insert_test_logic
```

Related Commands

add_control_points

delete_test_points

report_test_points

set_test_point_insertion_options

add_output_masks

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

Mode: setup

Usage

add_output_masks ***primary_output***… | **-All**

For dft -scan

add_output_masks {***primary_output***… | **-All**} [-Nohold | -Hold {0 | 1}]

Description

patterns -scan Details

Ignores any fault effects that propagate to the primary output pins you name.

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command allows you the ability to flag primary output pins that do not have strobe capability. The tool classifies the faults whose effects only propagate to that observation point as ATPG_untestable (AU).

dft -scan Details

The tool uses primary output pins as the observe points during the scan identification process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the identification process.

There are specific design practices that mask specific primary output pins in order to group the sequential elements associated with those primary outputs (for example, if you are using wrapper chains). For more information on wrapper chains, refer to "Understanding Wrapper Chains" in the *Tessent Scan and ATPG User's Manual*.

You can set a default mask for all output and bidirectional pins using the set_output_masks command. You can add a hold value to a default mask with the add_output_masks command, or remove a hold value using the delete_output_masks command. To turn off the default masks for all output pins, you must use the set_output_masks command with the Off literal.

Arguments

- ***primary_output***

  A repeatable string that specifies the name of the primary output pin you want to mask.

- **-All**

  A switch that specifies to mask all primary output pins.

- -Nohold **(dft -scan)**

  An optional switch that allows any state value on the specified primary output pin. This is the default.

- -Hold 0 | 1 **(dft -scan)**

  An optional switch and literal pair that specifies the state to maintain on the primary output pin. If you use this switch, you can explicitly specify that the output pin is driving the inactive value. The choices for the hold literal value are as follows:

  0 — Maintains a low logic state on the primary output pin.

  1 — Maintains a high logic state on the primary output pin.

Example 1

The following example specifies the primary output pins that will not have the strobe capability on the hardware tester:

```
add_output_masks qb1 qb2 qb3
```

Example 2

The following dft -scan example specifies the primary output pins that the add_output_masks command associates with the wrapper cells.

```
add_output_masks out1 out2 out3
```

When you issue the analyze_wrapper_cells command later in the session, the toolidentifies all the sequential elements that are observable only through the masked primary output pin. Then, when you issue the insert_test_logic command, the tool stitches all those scan cells it previously identified as being wrapper cells into one wrapper chain.

Related Commands

analyze_restrictions
delete_output_masks
report_output_masks
set_output_masks

add_primary_inputs

Context: all_contexts

Mode: setup

Usage

add_primary_inputs ***net_pathname***... [-internal] [-name *user_name*]

Description

Adds a primary input (PI) to the specified nets or pins.

This command also designates them as user-class PIs rather than system-class PIs described in the original netlist. Use the -internal switch to disconnect the original drivers of the net, so that the added PI becomes the only driver of the net or pin; otherwise, the net is treated as a wired net if there are other drivers besides the newly added PI. Use the -name switch to specify a list of internal pin pathnames to connect to a single, new primary input pin. You can display the user-class, system-class, or full-class PIs by using the report_primary_inputs command.

Arguments

- ***net_pathname***

  A required, repeatable string that specifies the pathname of a net or pin to which you want to add a primary input. If the -internal switch is used, net_pathname lists internal net pathnames. If both the -internal and -name switches are used, net_pathname lists internal net or pin pathnames to merge into a single, new primary input pin. The net_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -internal

  An optional switch that specifies net_pathname is an internal net or pin. This switch disconnects the original drivers of the net or pin, making the added primary input the only driver of the net or pin. These primary inputs are not added to the top-level interface when saving patterns. Use this switch to define internal control signals that originate from internal circuitry.

Note

The tool does not fault PIs created using the -internal switch. This allows interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

Note

When you use the write_patterns -Mode_internal switch when saving serial or parallel simulation patterns, the test bench still forces the internal nodes on net_pathname even though the corresponding PIs are not included in the top-level interface. When you save the patterns using the -Mode_external switch, the test bench will *not* force those internal nodes. In this case, unless the circuit generates values on the internal nodes corresponding to what ATPG assumed was forced on the corresponding PIs, you may get simulation mismatches.

- -name *user_name*

  An optional switch and string pair that specifies the name for a new primary input pin name that will drive all the internal pins specified with the net_pathname argument.

Example 1

The following example adds two new primary inputs to the circuit and places them in the user class of primary inputs:

add_primary_inputs indata2 indata4

Example 2

The following example adds a PI to the net originally driven by the output of a NAND gate, disconnecting the output of the NAND from the net. This happens within the tool only; the tool does not modify the original source netlist. The example then reports on the new PI and the original driver of the net. The example begins in setup mode. Notice that the tool reassigns gate IDs in the modified internal netlist.

set_gate_level primitive
create_flat_model
report_gates /ix157/Y

```
//  /ix157 (94)  NAND
//      A0     I  25-/reg_0_/Q
//      A1     I  33-/reg_1_/Q
//      Y      O  142-/d_2_mult_0_/A1
```

add_primary_inputs -internal /d_2_mult_0_/A1

```
//  Warning: Flattened model deleted.
//  Warning: Primary input is added at net '/nx16'
```

report_primary_inputs /nx16

```
USER:  /nx16 CUT
```

set_system_mode analysis
report_gates /nx16

```
//  /nx16 (15)  PI
//      nx16  O  /d_2_mult_0_/A1
//
//  User-added pin driven by "/ix157/Y" (95) in the source netlist.
//
```

report_gates /ix157/Y

```
//  /ix157 (95)  NAND
//      A0     I  26-/reg_0_/Q
//      A1     I  34-/reg_1_/Q
//      Y      O
//
//  Gate drives user-added pin "/nx16" (15) in the source netlist.
//
```

Example 3

The following example connects internal nets “xyz/abc/pina xyz/def/pina xyz/ghi/pina” to a single, new primary input and disconnects the original drivers of the nets, making the added primary input the only driver of the nets. The name of the new primary input pin is “group_pi”.

add_primary_inputs -internal xyz/abc/pina xyz/def/pina xyz/ghi/pina -name group_pi

Related Commands

add_clocks
add_primary_outputs
delete_primary_inputs
report_primary_inputs

add_primary_outputs

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

Mode: setup

Usage

add_primary_outputs ***net_pathname***... [-internal] [-name *user_name*]

For dft -scan and dft -test_points

add_primary_outputs ***net_pathname***… [-internal] [-name *user_name*]

Description

Adds primary outputs to the specified net.

The add_primary_outputs command adds an additional primary output to each specified net or pin. Once added, the tool defines them as user-class primary outputs. The tool defines the primary outputs described in the original netlist as system class primary outputs. You can display the user class, system class, or full classes of primary outputs using the report_primary_outputs command.

Arguments

- ***net_pathname***

  A required, repeatable string that specifies the net or pin to which you want to add primary outputs. The net_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -internal

  An optional switch that specifies net_pathname is an internal net or pin that is driven by the driver of the net_pathname. These primary outputs are not added to the top-level interface when saving mode_external patterns, such as STIL and WGL.

- -name *user_name*

  An optional switch and string pair that specifies the name for a new primary input pin name that will drive all the internal pins specified with the net_pathname argument.

Note

The tool does not fault POs created using the -internal switch. This allows interchange of fault lists between runs of the tool across different modes where you may have used different cut points.

Example

The following example adds a new primary output to the circuit and places it in the user class of primary outputs:

```
add_primary_outputs outdata1
```

Related Commands

add_primary_inputs
delete_primary_outputs
report_primary_outputs

add_processors

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion (dft -edt context only)

Usage

add_processors {***hostname***[:{*cpus* | MAXcpu}]}…

Description

Enables the tool to run multiple processes in parallel on multiple machines to reduce runtime.

The add_processors command enables the tool to use multiple processors simultaneously on specified host machines when executing multiprocessing commands for ATPG or fault simulation. For more information, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

Running multiple processes in parallel can sometimes reduce runtime significantly, especially for large designs. Be sure to load your design in the tool before using the add_processors command. This allows the tool to more accurately estimate memory requirements for the added processors.

Without access to multiple processors, the tool executes as a single master process. When you make additional processors available with the add_processors command, there will still be a single master process, but the tool starts up additional slave processes for parallel execution and coordinates communication between processes.

During ATPG and fault simulation, the tool can optionally use multiprocessing when executing the following commands:

- compress_patterns (fault simulation)
- create_patterns (ATPG)
- identify_redundant_faults (ATPG)
- order_patterns (fault simulation)
- simulate_patterns (fault simulation)

All multiprocessing commands alert you with a message when the design is being sent to slave processors. This message may not appear after subsequent commands if the other processors have already received the design and you have issued no commands in the interim that would affect the information the other processors received. There is currently a limit of 1024 processors per add_processors command.

Tip: To stop the command from waiting any longer for requested slave processes not yet received, and proceed with those it has, use the Control-C key sequence.

Note

IPv4 address support differs slightly by platform due to differences in the underlying system call implementations. These differences may be machine dependent due to dynamic linking with libraries resident on the host computer.

Arguments

- ***hostname***

  A required, repeatable string or literal that specifies a host machine or job scheduler that the tool can use for slave processes.

  The hostname choices are as follows:

  machine_name — A string that specifies a host machine by its network name.

  LOCALHOST — A literal that specifies to use the machine you invoked the tool on as the host machine.

  IPaddress — A string that specifies a host machine by its IP address.

  LSF — A literal that specifies to use available Load Sharing Function (LSF) host machines.

  SGE — A literal that specifies to use available Sun Grid Engine (SGE) host machines.

  GENERIC — A literal that specifies to use a host machine as determined by a custom job scheduler. This option requires that you first issue a set_multiprocessing_options command with the generic_scheduler argument to inform the tool how to launch the custom job scheduler.

Note

The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. See "Multiprocessing Requirements" in the *Tessent Scan and ATPG User's Manual* for more information.

- :{*cpus* | MAXcpu}

  A colon (:) followed by a positive integer or the MAXcpu keyword with which you can optionally specify the number of processes on the specified host machine or job scheduler to use as slave processes. You can specify up to 32 processes per host. If you specify 0 processes, the tool issues a warning.

  If you specify a positive integer, the tool uses that number of processes.

  If using a job scheduler, the tool uses as many processes as the scheduler provides in the time allotted. The time allotted is determined by the setting of the scheduler_timeout variable, which you can change using the set_multiprocessing_options command.

  If you specify the Maxcpu literal, the tool automatically determines the number of processes the host can accommodate (less one if the host must also run the master process) and use up to that number of processes as slave processes. The invocation default for SGE, LSF and generic is four processes; for the others it is one.

Note

The Maxcpu literal does not apply to grid scheduling or generic scheduling. The tool's Maxcpu calculations are based on currently running processes without considering other slave process specifications on the same command line. If, for example, you use the Maxcpu literal twice in the same command for a given host, the number of slave processes on that machine could exceed the number of available processors.

Example 1

The following example adds processors to the master host, which increases the thread count instead of adding additional slaves:

> **add_processors localhost:4 ;** # adds 4 threads to the master process for a total thread count of 5

```
//   Adding 4 threads to birdeye (master)
// Master with 5 threads running.
```

> **report_processors**

```
//  hosts             threads   arch      CPU(s)    %idle   free RAM  process size
//  ----------------  -------  ------  -----------  -----  ---------  ------------
//  birdeye (master)        5  x86-64  8 x 2.9 GHz    89%  113.91 MB     232.56 MB
//  master with 5 threads running.
```

Example 2

The following example starts a distributed slave process on the host "odin" that runs with 4 total threads. The add_processors command can start only one process per slave host, and create additional processors (to master or slave) as additional threads.

> **add_processors odin:4**

```
//    Adding 4 threads to odin (new slave)
//  Master with 1 thread and 1 slave with 4 threads running (5 total threads).
```

Related Commands

compress_patterns
create_patterns
delete_processors
identify_redundant_faults
order_patterns
report_processors
set_multiprocessing_options

add_read_controls

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

Mode: setup

Usage

add_read_controls **0** | **1** ***primary_input_pin***...

Description

Adds an off-state value to specified RAM read control lines.

The add_read_controls command defines the circuit read control lines and assigns their off-state values. The off-state value of the pins that you specify must be sufficient to keep the RAM outputs stable. You cannot use clocks, constrained pins, or equivalent pins as read control lines.

Note: The tool automatically adds a read control to the clock list during flattening if the flattening process reaches the clock pin of an edge-triggered RAM. In this case, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0**

  A literal that specifies 0 as the off-state value for the RAM read control lines.

- **1**

  A literal that specifies 1 as the off-state value for the RAM read control lines.

- ***primary_input_pin***

  A required, repeatable string that lists the primary input pins you want to designate as RAM read control lines, and to which you are assigning the given off-state value.

Example

The following example assigns an off-state value of 0 to two read control lines, r1 and r2:

```
add_read_controls 0 r1 r2
set_system_mode analysis
create_patterns
```

Related Commands

analyze_control_signals
delete_read_controls
report_read_controls

add_register_value

Context: all contexts

Mode: setup

Usage

add_register_value ***value_name***
{[*value_string* [-Dynamic] [-Radix {Binary | Decimal | Octal | Hexadecimal}]
{-Width [*integer*]}
| {-Pattern_count { SCan_test | CHain_test | ALl }} }
[-LSB_shifted_first] [-INput_pin_inversion] [-OUtput_pin_inversion]

Description

Creates a static or dynamic variable by allowing you to specify a register identifier, ***value_name***, and, optionally, the state string, ***value_string***.

When you specify a value_string, the variable is static by default unless you use the -Dynamic switch.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- ***value_name***

  A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.

- *value_string*

  An optional state string whose default radix is binary. When you specify a value_string, the variable is static by default unless you use the -Dynamic switch. Note that you aren't allowed to specify a value_string when you use the -Pattern_count switch.

- -Dynamic

  An optional switch that forces the variable to be dynamic when you provide an optional value_string. By default, when you provide a value_string, the variable is static.

- -Radix {**Binary** | Decimal | Octal | Hexadecimal}

  An optional switch and literal that specifies the radix of the value_string. If you specify the -Radix switch, then you must specify the register width using the -Width switch.

- -Width *integer*

  A switch and integer pair that specifies the register width. When defining a dynamic variable, you must use this switch. If you do not know the actual width value until after test generation completes, then use the -Width switch without ***integer***, and specify the width using the set_register_value command.

The maximum width of a register value is 1,000,000,000. A specified width larger than this produces an error.

- -Pattern_count {SCan_test | CHain_test | ALl}

  A switch and literal pair that specifies using the total number of patterns generated by the tool when the variable is dynamic. Note that when you use this switch, you're aren't allowed to specify a value_string. Choose from one of the following:

  SCan_test — Scan patterns.

  CHain_test — Chain patterns.

  ALl — Both scan and chain patterns.

- -LSB_shifted_first

  An optional switch that specifies shifting the less significant bit first into the register.

- -INput_pin_inversion

  An optional switch that specifies an inversion on the input pin when loading the register.

- -OUtput_pin_inversion

  An optional switch that specifies an inversion on the output pin when unloading the register.

Example

The following example deletes all register value variables and then specifies two dynamic register value variables with a decimal radix and register width of 37.

delete_register_value -all

add_register_value setup_TDI1 0 -dynamic -radix decimal -width 37

add_register_value setup_TDI2 10 -dynamic -radix decimal -width 37

report_register_value

```
//  Register value name                value string                width
//  -------------------  -------------------------------------  -----
//       setup_TDI1      0000000000000000000000000000000000000   37
//       setup_TDI2      0000000000000000000000000000000001010   37
```

Related Commands

delete_register_value
report_register_value
set_register_value

add_scan_chains

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

Mode: setup

Prerequisites: You must define the scan chain group with the add_scan_groups command prior to using this command.

Usage

add_scan_chains [-Internal | -Load_only] {***chain_name group_name input_pin output_pin***}...

For “dft -edt” context

add_scan_chains [-Internal] {***chain_name group_name input_pin output_pin***}...

Description

Specifies a name for a pre-existing scan chain within the design.

The add_scan_chains command defines a scan chain that exists in the design. A scan chain references the name of a scan chain group, which you must define prior to issuing this command.

A pre-existing scan chain is a serially connected set of scan cells that has been stitched together in a previous scan insertion operation, or that you created manually. In order for the tool to be able to correctly insert scan elements in the remainder of the design or to run rules checking on existing scan circuitry, you need to notify the tool of this circuitry, so it can treat those elements in the chain differently than the elements in the rest of the design.

When you use this command to notify the tool of a pre-existing scan chain, dft -scan context removes the scan cells in the scan chain from the eligible scan elements list. Because the tool removes the scan cells in the scan chain from the scan candidate list, the rules checker does not perform the usual scannability checks on those scan cells. However, the rules checker does perform additional rules checking based on the declaration of pre-existing scan elements that pertain to the checking of the validity and the operation of that scan chain.

If the design does have a pre-existing scan chain that you declare with the add_scan_chains command, you need to provide information on the operation of that scan chain in a test procedure file.

Arguments

- ***chain_name***

  A required repeatable string that specifies the name of the scan chain to add to the specified scan group.

- ***group_name***

  A required repeatable string that specifies the name of the scan chain group to add the specified scan chain to.

- ***input_pin***

  A required repeatable string that specifies the input pin of the specified scan chain.

- ***output_pin***

  A required repeatable string that specifies the output pin of the specified scan chain.

- -Internal

  An optional switch that defines internal scan chains.

Note

The -Internal switch specifies compressed scan chains when using Tessent TestKompress with EDT logic. Do not use it to define uncompressed scan chains (chains connected directly to a primary input and primary output and not driven by or observed through the Tessent TestKompress logic). See the "Scan Chain Insertion" section in the *Tessent TestKompress User's Manual*.

- -Load_only **(Not supported in the hybrid TK/LBIST flow)**

  An optional switch that specifies the definition of a load-only scan chain which is an uncompressed scan chain in which the scan input is a primary input and the scan output is an internal pin that is only used for the purpose of scan chain tracing, A load_only scan chain does not need to be unloaded which saves chip-level pins and scan data.You can specify a load_only scan chain for special cases such as on-chip clock controllers that only need to be loaded but do not need to be unloaded.

  The tool automatically adds OX constraints on all cells of the chain using the chain masking mechanism, since those cells will not be unloaded and observed. In the generated patterns or mode_external testbench, the internal scan chain outputs are ignored. These chains can be added with other normal scan chains, and/or with EDT blocks. However, the load-only chains must be uncompressed and loaded only from ports, not a decompressor.

Note

This switch does not support mapping EDT patterns to single bypass chain mode. Also, just like normal uncompressed chains, load-only uncompressed chains should not be added during the EDT IP phase.

Example 1

The following example defines two scan chains (chain1 and chain2) that belong to the scan group (grp1):

```
add_scan_groups grp1 atpg.testproc
add_scan_chains chain1 grp1 si1 so1
add_scan_chains chain2 grp1 si2 so2
```

Example 2

The following example shows add_scan_chains commands similar to those Tessent TestKompress would write in the **_edt.dofile* dofile, during the IP Creation Phase, for the two

scan chains of the preceding example. By default, the top level module name "/my_design" is prefixed to the pin names.

```
add_scan_groups grp1 *_edt.testproc
add_scan_chains -internal chain1 grp1 /my_design/si1 /my_design/so1
add_scan_chains -internal chain2 grp1 /my_design/si2 /my_design/so2
```

Related Commands

add_scan_groups
delete_scan_chains
report_scan_chains

add_scan_groups

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

Mode: setup

Usage

add_scan_groups {***group_name test_procedure_filename***}...

Description

Specifies a group name for a set of pre-existing scan chains in a design. Only one group name can be specified per session.

If the design has pre-existing scan chains, those scan chains must belong to a scan group. The add_scan_groups command also specifies the corresponding test procedure file for that scan group. A sequence of procedures in the test procedure file defines the operation of the scan chains within a scan group. The scan group must have a corresponding test procedure file.

When needed, you can also use the add_scan_groups command to provide initialization values for non-scan memory elements. If the design does not have pre-existing scan chains and you need to initialize some non-scan memory elements for scannability checking, you can specify “dummy” as the group_name along with a test_procedure_filename. Then, within the test procedure file, you can use the optional **test_setup** procedure to initialize the non-scan memory elements.

As a result, in dft -scan context, the tool models the non-scan memory elements as a TIE1 or TIE0.

Note

You must also define these same elements as non-scan using the add_nonscan_instances command.

Arguments

- ***group_name***

  A required string that specifies the name of the scan chain group that you want to add to the system.

- ***test_procedure_filename***

  A required string that specifies the name of the test procedure file that contains the information for controlling the scan chains in the specified scan chain group.

Example

The following example defines a scan chain group, group1, which loads and unloads a set of scan chains, chain1 and chain2, by using the procedures in the file, *scanfile*:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 testout2
add_scan_chains chain2 group1 indata4 testout4
```

Related Commands

add_scan_chains
delete_scan_groups
report_scan_groups
read_procfile
write_procfile

add_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Usage

add_scan_instances ***pathname***… | ***instance_expression***
[-INStance | -Module]

Description

Adds specified instances to the scannable instance list.

When inserting scan chains, the add_scan_instances command ensures that the tool includes specific sequential instances in the scan list. The tool generates the scan list during the scan identification process. If the specified instance does not pass the rules checking process, it is not included in the identified scan list.

You can use the delete_scan_instances command to remove the exception status that you have attached to an instance with the add_scan_instances command. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

Arguments

- ***pathname***

  A required repeatable string that specifies the pathnames of the sequential instances to add to the scan instance list. If the specified instance is hierarchical, then all sequential instances beneath it are also added to the scan list.

- ***instance_expression***

  A required string representing a list of instances within the design. The string instance_expression is defined as:

  { string | string * } ...

  The asterisk (*) can be used as a wildcard character. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. If you use a pathname expression to select several instances, the tool will add_scan_instances for all the pins on those instances; but if the expression specifies a location below the instance level of an ATPG library model, an error message displays.

- -INStance | -Module

  An optional switch that specifies whether the specified pathnames are instances or modules. The default is -INStance.

Examples

The following example adds two sequential instances to the identified scan list and then runs the scan identification process.

```
add_scan_instances i_1006 i_1007

insert_test_logic -number 10
```

Related Commands

delete_scan_instances

report_seq_transparent_procedures

add_scan_models

Context: dft -scan, dft -test_points

Mode: setup

Usage

add_scan_models ***model_name***…

Description

Specifies that the tool is to flag every instance of the named DFT library model for inclusion into the identified scan list.

If the tool is inserting scan chains, then the add_scan_models command ensures the tool includes all instances of a specific sequential DFT library model in the scan list. The tool generates the scan list during the scan identification process, which you initiate with the insert_test_logic command in the analysis mode. The tool subsequently replaces all instances in the scan list with the equivalent DFT library scan model during scan synthesis, which you initiate with the insert_test_logic command while you are in analysis mode.

If the tool is in the analysis mode, then the add_scan_models command only flags the scannable instances of the specified library model for inclusion into the scan list. Scannable instances are those sequential instances previously passing all the scannability checks run by the Design Rule Checker when you placed the tool into the analysis mode. The tool omits library model instances from the scan list if the instance fails the design rule checks.

The add_scan_models command flags all instances of the specified sequential DFT library model, where the add_scan_instances command only individually flags sequential design instance(s).

Arguments

- ***model_name***

  A required, repeatable string specifying the model names that you want to add to the scan model list. Enter the model names as they appear in the DFT library.

Examples

The following example flags all instances of the specified DFT library model for inclusion into the scan list.

```
add_scan_models dff1a
set_system_mode analysis
insert_test_logic -number 10
```

Because of the previous setup in this example, when the tool runs the scan identification process, it chooses the optimal 50 percent of eligible scan instances, ensuring that it includes all eligible instances of the dff1a model in that 50 percent of identified scan instances.

Related Commands

delete_scan_models

report_scan_models

add_scan_partition

Context: dft -scan

Mode: setup, analysis

Usage

add_scan_partition ***object_name*** **-INstance** {***pathname...*** | ***instance_expression...***} **-MOdule** ***module_name...*** **-LIbrary_model** ***library_model_name...*** [-NUmber *integer* | -MAx_length *integer*] [-EDT] [-VErbose]

Description

Specifies a grouping of scan cells (a partition) in which scan chains are inserted separately from the remaining scan cells in the design.

The remaining cells are placed into a default scan partition called default_scan_partition. If a module instance name is supplied in the list of instances, all sequential instances hierarchically under this given module instance are added to the scan partition. If a module name is supplied, all instances of that module and the sequential instances hierarchically under these instances are added to the scan partition. If a library model name is supplied, all instances of this library model are added to the scan partition if this library model is defined for a sequential instance. At least one of the -INstance, -MOdule, and -LIbrary_model switches should be supplied. They can also be supplied together in the same command line.

It is advisable to add scan partitions in Setup mode, especially when the partitions require dedicated scan enable signals that, preferably, should be defined in Setup mode or when a power data is read in into the design. In the case of power data, the specified power domains are mapped into scan partitions during transition from Setup mode to analysis mode; so, it is important to make sure that there are no conflicts between the power domains and user-defined scan partitions.

The specified group of cells is placed in scan chains based on the number of chains or the maximum chain length option specified with this command, along with the -clock/-edge merge options of the insert_test_logic command. The -number and -max_length options of the insert_test_logic command apply to the entire design as follows:

- The -number value specified by add_scan_partition cannot exceed the -number value specified by insert_test_logic; if -number is not specified by add_scan_partition, a single chain is inserted for the partition by default.
- If the -max_length value is not specified by add_scan_partition, the -max_length value of insert_test_logic is used to generate the partition's scan chains.

Note

You can specify either a number chain specification or a maximum length chain specification in a single session among the insert_test_logic, add_scan_partition, and set_wrapper_chains commands, but, you cannot specify both. For example, if you specify the number of chains for a scan partition using the add_scan_partition -number option, you cannot use the insert_test_logic -max_length option for the remainder of the scan cells.

The tool generates a single dofile and testprocedure file even if there are multiple scan partitions defined. However, in the dofile, the scan chain declarations of the same scan partition are separated from the other groups of scan chain declarations by a comment line which indicates the scan partition name. Also, the groups of scan chain declarations are in the same order as the defined scan partitions. If the -Edt switch is used, the tool writes out the Tessent TestKompress command, “Add Edt Block name”, before every group of scan chain declarations, where name is the name of the scan partition specified by object_name.

Default scan pin naming for each partition cannot be specified by means of the set_scan_pins command. However, individual scan pin naming per chain can be specified by means of the add_scan_pins command. This gives the control over scan pin naming for each scan partition since scan chains are created in the order the scan partitions are defined and the number of scan chains for each scan partition is known.

Note

The scan cells of subchains cannot be included in scan partitions. This is because a subchain container can be a blackbox instance, the sequential cells of which do not exist in the netlist. Therefore, subchains can be included in scan partitions by specifying the pathname of their container module instances, or the module name of the container modules.

Arguments

- ***object_name***

  A required string that specifies a name for the scan partition.

- **-INstance** {***pathname...*** | ***instance_expression...***}

  A required switch and a repeatable string that specify the pathname(s) of the sequential instances that you want to place into the scan partition. If a module instance pathname is specified, all sequential instances hierarchically under that instance are added to the partition. This switch can be used along with the -Module and -Library_model switches (and their arguments) in the same command line. However, all three cannot be omitted.

  The repeatable string can be in the form of an absolute instance pathname or a pathname in regular expression form. This argument does not support pathnames to objects below the instance level of an ATPG library model. If the instance pathname/expression cannot be

mapped to any sequential element in the design, the tool generates an error message. The string instance_expression is defined as:

{string | string *} ...

The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

- **-MOdule *module_name...***

  A required switch and a repeatable string that specify the module name(s) of the instances that are the containers of the sequential instances that you want to place into the scan partition. All sequential instances hierarchically under these container instances are added to the partition. This switch can be used along with the -Instance and -Library_model switches (and their arguments) in the same command line. However, all three cannot be omitted.

- **-LIbrary_model *library_model_name...***

  A required switch and a repeatable string that specify the library model name(s) of the sequential instances that you want to place into the scan partition. This switch can be used along with the -Instance and -Module switches (and their arguments) in the same command line. However, all three cannot be omitted.

- -NUmber *integer*

  An optional switch and integer pair that specify the exact number of scan chains that you want the tool to insert for the scan partition specified. Final results depend upon the number of scan candidates. The default number of chains is 1.

Note

The number of scan chains specified with insert_test_logic -number applies to the entire design. The number of scan chains specified with other commands such as set_wrapper_chains and add_scan_partition is subtracted from this number.

The total number of all scan chains individually specified with the set_wrapper_chains and add_scan_partition commands cannot exceed the number specified by the “insert_test_logic -number” command.

- -MAx_length *integer*

  An optional switch and integer pair that specify the maximum number of scan cells that the tool can stitch into each new scan chain; this argument overrides the -max_length value of the insert_test_logic command which is applied to the entire design. The tool evenly divides the scan cells into scan chains the length of which does not exceed the -max_length value of the add_scan_partition command; if -max_length is not specified, the -max_length value of the insert_test_logic command is used, if provided; otherwise, a single chain is inserted for the partition by default. Final results depend upon the number of scan candidates.

- -EDT

  An optional switch that specifies the tool to write out the Tessent TestKompress command "Add Edt Block name" before each group of scan chain declarations in the dofile written out by the tool. In the dofile, the scan chain declarations of the same scan partition are grouped together and separated from the other groups of scan chain declarations by a comment line. The "name" string in the Tessent TestKompress command is the same as the scan partition name specified by object_name.

- -VErbose

  An optional switch that turns on verbose transcript printing. When specified, the pathnames of the sequential cells included in the scan partition are printed in the session transcript.

Examples

In the following example, two scan partitions are defined. The first partition, partA, is defined using exact pathnames of the sequential instances, and the second partition, partB, is defined by the container module instance of the sequential instances. A single scan chain is inserted by default for partA, whereas two scan chains are inserted for partB. Two scan chains are inserted for the remaining cells in the default scan partition, as specified by the -number argument of the insert_test_logic command. The -Edt switch is used for partA, allowing the tool to write out the Tessent TestKompress command, Add Edt Block, in the dofile, in addition to the scan chain declarations. Also, Individual scan I/O pins per chain and default scan I/O pin naming are specified by means of the add_scan_pins and set_scan_pins commands, respectively.

```
add_clocks 0 clk1
add_scan_partition  partA  -instance udff1 umodA/udff32 umodB/udff5 -edt   # 1 chain
add_scan_partition  partB  -instance umodC  -number 2 -edt   # 2 chains

add_scan_pins  chain1  partA_chain1_si  partA_chain1_so
add_scan_pins  chain2  partB_chain1_si  partB_chain1_so
add_scan_pins chain3 partB_chain2_si  partB_chain2_so
set_system_mode analysis
set_scan_pins  input  -indexed  -prefix mysi  -initial 1
set_scan_pins  output  -indexed  -prefix myso  -initial 1
insert_test_logic -number 5     # 2 chains inserted for the default scan partition
write_atpg_setup fscan
```

So, a total of five scan chains are inserted. After mapping the individual scan I/O pins per chain, new scan I/O pin names are generated for the remaining chains based on the specified default pin naming. The dofile generated by the tool looks similar to the one below. Note that the scan chain declarations of each scan partition are denoted by a comment line. The order of the groups of declarations is the same as the order of the scan partition declarations. The default scan partition chains are printed as the last group. Also note that the Tessent TestKompress add_edt_blocks commands are written out since the -Edt switch is used. For the default scan partition, the tool generates the name "edt_top_block" to use it as the edt block identifier string in the add_edt_blocks command.

```
add_scan_groups grp1 fscan.testproc

# The scan chains of scan partition "partA"
add_edt_blocks partA
```

```
add_scan_chains chain1 grp1 partA_chain1_si  partA_chain1_so
# The scan chains of scan partition "partB"
add_edt_blocks partB
add_scan_chains chain2 grp1 partB_chain1_si  partB_chain1_so
add_scan_chains chain3 grp1 mysi1 myso1
# The scan chains of scan partition "default_scan_partition"
add edt block edt_top_block
add_scan_chains chain4 grp1 mysi2 myso2
add_scan_chains chain5 grp1 mysi3 myso3

add_clocks 0 clk1
```

Related Commands

delete_scan_partitions
insert_test_logic
report_scan_partitions

add_scan_pins

Context: dft -scan

Mode: setup, analysis

Usage

add_scan_pins ***chain_name scan_input_pin scan_output_pin***

Description

Declares the name of a scan chain at the top-level module and assigns the corresponding scan input pin, scan output pin, and optionally, scan clock pin you associate with the chain.

Tip: For information on assigning scan pins to scan chains and the fixed-order file, refer to "Naming Scan Input and Output Ports" in the *Tessent Scan and ATPG User's Manual*.

If you specify nonexistent signal names for the scan input or scan output pins, the tool generates a warning and adds the appropriate pins when you perform scan synthesis with the insert_test_logic command.

The tool uses the default names when generating the scan chain. The default name of the scan chain is chain1, with the corresponding scan input pin named scan_in1 and the corresponding scan output pin named scan_out1. If the tool inserts multiple scan chains, the default naming increments the numerical suffix of each argument by one for each chain, such as chain2, scan_in2, and scan_out2.

Each use of the add_scan_pins command specifies the naming for a single scan chain. You can optionally use the set_scan_pins command to globally set the default naming conventions for scan chains that do not use existing design pins.

Arguments

- ***chain_name***

  A required string that specifies the name of the scan chain with which you want the tool to associate the *scan_input_pin* and *scan_output_pin* names.

- ***scan_input_pin***

  A required string that specifies the scan input pin name of the scan chain. This pin can be a top-level input pin, top-level bidirectional pin, or an internal signal.

  In addition to a primary input pin name (for example, scan_in), you can also specify an internal instance pin pathname as the scan_input_pin value. If you do so, that pin pathname must be an output pin of an instance. For example, you could use an internal instance pin pathname "/I116/d", where "I116" is the instance name of the I/O cell and "d" is the pin name. If the specified internal instance pin cannot be traced back to a primary input pin through a simple path (only inverters or/and buffers), when the write_atpg_setup command is issued, a warning message is written to the top of the ATPG dofile and the following

command is added prior to the add_scan_chains command referencing the internal scan_input_pin argument:

add_primary_input -internal scan_input_pin

Example 1 shows the content written to the ATPG dofile in this case.

If the pin is a top-level bidirectional pin, the tool assumes that you configured the pin to operate as an input during the scan test and does not check for correct configuration.

- ***scan_output_pin***

  A required string that specifies the scan output pin name of the scan chain. This pin can be any of the following:

  - a top-level output pin
  - a top-level bidirectional pin (single driver)
  - an internal signal

Note

If the scan output pin you specify has a functional connection, the tool multiplexes this connection with the connection line from the last scan cell of the scan chain.

In addition to a primary output pin name (for example, scan_out), you can also specify an internal instance pin pathname (for example, "/I116/q") for the scan_output_pin value, providing this pin pathname is an input pin of an instance. If the specified internal instance pin cannot be traced forward to a primary output pin through a simple path (only inverters or/and buffers), when the write_atpg_setup command is issued, a warning message is written to the top of the ATPG dofile and the following command is added prior to the add_scan_chains command referencing the internal scan_output_pin argument:

add primary output scan_output_pin

Example 2 shows the content written to the ATPG dofile in this case.

If the pin is a top-level bidirectional pin, the tool assumes that you configured the pin to operate as an output during the scan test and does not check for correct configuration.

Example 1

The following example specifies an internal pin pathname that the tool cannot trace back to the primary input. See the resulting dofile contents in Example 7-1 on page 536.

```
add_clocks 0 clk

set_tristate_gating bus -control ten
set_tristate_gating out1 out4 -control ten
set_tristate_gating out3 out2
set_bidi_gating on

add_scan_pins c1 udff1/Q tbus2_drv1/A
add_scan_pins c2 bidi_1/X tbus1_drv2/A
```

```
set_system_mode analysis
report_dft_check -tristate
insert_test_logic -number 2
report_scan_chains
write_atpg_setup results/tri_on_ten -replace
write_design -output_file results/tri_on_ten.v -replace
```

Example 7-1. Generated dofile Tracing Back to Primary Input

```
# The generated dofile contains references to internal pins
# The file may require editing to make it function properly.
add_scan_groups grp1 results/tri_on_ten.testproc
add primary inputs -internal udff1/Q # internal input pin
add primary inputs -internal bidi_1/X # internal input pin
add_scan_chains c1 grp1 udff1/Q out3
add_scan_chains c2 grp1 bidi_1/X out2
add_clocks 0 clk
add_input_constraints test_en -C1
```

Example 2

The following example specifies an internal pin pathname that the tool cannot trace forward to the primary output. See the resulting dofile contents in Example 7-2 on page 537.

```
add_clocks 0 clk

set_tristate_gating bus -control ten
set_tristate_gating out1 out4 -control ten
set_tristate_gating out3 out2
set_bidi_gating on

add_scan_pins c1 bidi_2/X tbus2_drv1/A
add_scan_pins c2 bidi_1/X udff1/D

set_system_mode analysis
report_dft_check -tristate
insert_test_logic -number 2
report_scan_chains
write_atpg_setup results/tri_on_ten -replace
write_design  -output_file results/tri_on_ten.v -replace
```

Example 7-2. Generated dofile Tracing Forward to Primary Input

```
# The generated dofile contains references to internal pins
# The file may require editing to make it function properly.
add_scan_groups grp1 results/tri_on_ten.testproc
add_scan_chains c1 grp1 io3 out3
add primary output udff1/D # internal output pin
add primary output bidi_2/X # internal output pin
add_scan_chains c2 grp1 bidi_2/X udff1/D
add_clocks 0 clk
add_input_constraints test_en -C1
```

Related Commands

delete_scan_pins
report_scan_pins
set_scan_pins
write_atpg_setup

add_simdut_fault

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

Mode: setup, analysis

Usage

add_simdut_fault **-signal *signal_name* -fault [0 | 1]**

Description

Injects a stuck-at-0 fault or a stuck-at-1 fault into the specified signal for SimDUT simulation.

SimDUT uses the Verilog simulation capabilities to force the signal to 0 or 1. The Verilog capability has the following behavior: When you specify a pin pathname for the fault you want to inject, Verilog forces the associated signal to the specified value.

When you inject a fault at a specific pin pathname, SimDUT actually injects the fault at the whole net, not only on the specific pin. The fault acts like a fault injected on the upstream driver pin of the associated net. If you specify the input port of a cell and this port is attached to a net with fanout, then the fault acts like a fault injected on the driver port of that net.

When you execute a test, SimDUT simulates the injected fault as if it were an actual defect in silicon, collects the simulated failing cycles, and returns the results.

You can execute this command multiple times. When you specify a particular signal more than once, the last command that specifies the signal overrides all previous specifications. For example, if you first add a stuck-at-0 fault to signal A and later add a stuck-at-1 to signal A, SimDUT only adds the stuck-at-1 fault to signal A.

Arguments

- **-signal *signal_name***

A required switch and string pair that specifies the name of a signal from the Verilog netlist. Refer to "SimDUT Signal Naming Convention" in *Tessent SiliconInsight User's Manual for Tessent Shell* for more information

- **-fault [0 | 1]**

  A required switch and literal that specifies the fault type as follows:

  0 - Specifies stuck-at fault type SA0.

  1 - Specifies stuck-at fault type SA1.

Example

The following example injects an SAO fault into the SIMDUT_TB.CHIP.memory00.q_a signal.

```
add_simdut_fault -signal SIMDUT_TB.CHIP.memory00.q_a -fault 0
```

Refer to "Performing SimDUT Test Pattern Simulation" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a usage scenario.

Related Topics

delete_simdut_fault

execute_cdp_test

add_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

add_simulation_context ***context_name***

Description

Creates a new user-defined simulation context.

The add_simulation_context command allows you to create a new user-defined simulation context in which the tool can perform simulation and introspection. Initially, the tool propagates tied values, such as power and ground, and then sets the remaining simulation values to X. The command automatically sets the new simulation context to be the current simulation context.

Arguments

- ***context_name***

  A required string that specifies the name of the context. The context_name has to be different than one of the predefined context names listed in Table 7-40 on page 1648.

Example

The following example shows how to create a new simulation context:

```
SETUP> add_simulation_context myContext
//  'myContext' is the current simulation context.

SETUP> get_simulation_context_list -user_defined
{myContext}
```

Related Commands

copy_simulation_context
delete_simulation_contexts
get_current_simulation_context
get_simulation_context_list
report_simulation_contexts
set_current_simulation_context

add_simulation_force

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

add_simulation_force ***obj_spec*** -value ***value***

Description

Forces one or more gate_pin objects to the specified value.

The add_simulation_force command forces the specified gate_pin object(s) to the specified force value within the current simulation context. The actual simulation is not triggered until you issue the simulate_forces or simulate_clock_pulses command. If a specified gate_pin object has already been forced by an earlier add_simulation_force command, and the tool hasn't yet simulated the previous force, the new force value overwrites the existing one in case of a conflicting value. Note that adding a simulation force also updates the gate_pin attribute named simulation_value.

This command is allowed for user-defined simulation contexts only.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects.

- -value ***value***

  A required switch and value pair that specifies the force value applied to the specified gate_pin objects. The value is allowed to be one of the following: X, 0, 1, or Z.

Example

The following example shows the effect of adding a force on a particular gate_pin object, and then triggering the tool to simulate that force:

ANALYSIS> **set_current_simulation_context myContext**
ANALYSIS> **add_simulation_force [get_gate_pins I1] -value 1**
ANALYSIS> **report_simulation_forces**

```
Value   Object
------  ------
1       I1
```

ANALYSIS> **get_simulation_value_list [get_gate_pins I1]**
{ X }

ANALYSIS> **simulate_forces**
ANALYSIS> **get_simulation_value_list [get_gate_pins I1]**
{ 1 }

Related Commands

delete_simulation_forces
get_simulation_value_list
report_simulation_forces
simulate_clock_pulses
simulate_forces

add_sub_chains

Context: dft -scan, dft -test_points

Mode: setup

Usage

add_sub_chains ***object_name subchain_name scan_input_pin scan_output_pin length scan_type* [-Module | -Library_model | -Instance]** [-TEN *test_enable_pin* {0 | 1}] [-NO_Reordering] [-TYpe {None | Core | Input_wrapper | Output_wrapper}] [-EXTest_enable_pin *pinname*] [-Lockup_cell_at_output]

Description

Specifies a pre-existing scan chain (subchain) within a module, library model, instance, blackbox, or empty module in a hierarchical design to be stitched into the top-level scan chain during the stitching process.

Note

An empty module is a module defined in the netlist without any content; that is, it consists of only the input and output port definitions.

The tool checks the traceability of an added subchain like any other pre-existing scan chain if the subchain's container is an instance of a netlist module, is not blackboxed, and a test procedure file containing a shift and load_unload procedure is present. If a test procedure file is not present, the following warning message is issued and the modules where the added subchains reside are treated as black boxes:

```
// Warning: Unable to find load_unload and shift procedures: subchains
// tracing will not be done. Subchain containers will be treated as black
// boxes.
```

Any sequential cell inside a traceable subchain's container that is not a part of this subchain is treated as any other sequential cell in the design.

The scan_input port of a subchain must be either unconnected or tied to 0 or 1 or connected to the scan_output port of this subchain either directly or via buffers. During the scan chains stitching, the latter loop is broken; any buffers of the original loop are retained in the scan_output connection. The scan_output port of a subchain must be either unconnected or connected to the scan_input port of this subchain either directly or via buffers.

The scan_enable port, specified in scan_type, can be either unconnected or tied to 0 or 1.

Every subchain specification in Mux-DFF scan architecture must be accompanied by the subchain's clock definition (see add_subchain_clocks).

Every subchain is subject to S-rules checking during DRC.

During the block-by-block test logic insertion in the bottom-up design flow, the tool can write the subchain setup information to a file with the write_subchain_setup command at the lower design levels. This setup file, containing the add_sub_chains command for every top-level scan

chain at the lower design level, can be used as part of the setup at the higher design level to have the pre-existing lower design level scan chains (subchains) stitched into the new top-level scan chains.

Arguments

- ***object_name***

  A required string that specifies either the pathname of an instance, the name of a module, or the name of a library model. The object_name is the container where the subchain resides.

- ***subchain_name***

  A required string that specifies a name for the scan subchain. A unique subchain name should be used for each add_sub_chains command.

- ***scan_input_pin***

  A required string that specifies the scan input pin of the scan subchain.

- ***scan_output_pin***

  A required string that specifies the scan output pin of the scan subchain.

- ***length***

  A required integer that specifies the number of scan cells in the scan subchain.

- ***scan_type***

  Scan type usage:

  {Mux_scan {*scan_enable* [INVerted]}} |
  Clocked_scan *scan_clock* | Lssd *master_clock slave_clock*

  A required literal and multiple argument option that specifies the scan type and control of the scan subchain. Options include:

  - **Mux_scan *scan_enable*** [INVerted]
    Required switch, string, and literal that specifies the type, name, and internal inversion of the scan enable pin on the subchain container (module or library_model).

  - **Clocked_scan *scan_clock*** — A required literal and string pair that specifies the clocked-scan style of scan cells and the name of the scan clock for the scan subchain.

  - **Lssd *master_clock slave_clock*** — A required literal and two-string triplet that specifies Level-Sensitive Scan Design and the names of the master and slave clocks, respectively, for the scan subchain.

- -Module | -Library_model | -Instance

  An optional switch that specifies the type of the subchain container specified by the object_name argument. The container can then be a module (-Module), a library model (-Library_model), or an instance of a module/model (-Instance). The default type is -Module.

- -TEN *test_enable_pin* {0 | 1}

  An optional switch, string, and literal pair that specifies the test enable pin added in the sub-module that you want connected to the corresponding pin in the top module. The active value can be 0 or 1.

- -NO_Reordering

  An optional switch that specifies to keep the order of cells on the specified subchain's scan path intact when writing out the scanDEF file by placing them into an ORDERED section of the corresponding scan chain. When this switch is not specified, the subchain's sequential cells are written into a FLOATING section of the corresponding scan chain which allows them to be reordered.

- -TYpe {None | Core | Input_wrapper | Output_wrapper}

  An optional switch and a literal that specify the type of scan chain the subchain should be stitched into.

  This information is used to override the automatic identification of the scan chain type for this subchain's stitching into either a Core type, Input wrapper type, or Output wrapper type scan chain. The Input_wrapper and Output_wrapper types are applicable only in the wrapper chains flow triggered by the analyze_wrapper_cells command.
  When this switch is not specified, the tool automatically determines the scan chain type of a traceable subchain whenever possible (see the -INClude_sub_chains {ON | OFF} option of set_wrapper_chains). When this switch is specified with the explicit Input_wrapper, Output_wrapper or Core type, and the analyze_wrapper_cells command has been issued, the subchain is stitched into the appropriate Input, Output or Core chain. In any case, the subchain will be controlled by a scan enable signal appropriate to the type of the scan chain it is stitched into.

- -EXTest_enable_pin *pinname*

  An optional switch and string pair that specifies the extest enable pin on the subchain container module boundary. This switch is only valid when the -Type argument is specified as Wrapper because this pin is only used for wrapper type subchains.

- -Lockup_cell_at_output

  An optional switch that indicates when a lockup cell is present at the end of a sub chain (to avoid insertion of a new lockup cell during the stitching of sub chains at the top level in bottom-up flow). You can add this switch manually; the tool automatically adds this switch when writing out existing scan chains with the write_subchain_setup command.

Example 1

The following example defines a subchain on a sub-module:

```
add_sub_chains addr subc1 scan_in1 scan_out1 8 mux_scan  scan_en inverted
add_subchain_clocks subc1 0 ck -first_cell_clock -leading_edge
add_subchain_clocks subc1 0 ck -last_cell_clock -leading_edge
report_sub_chains
```

```
mux_scan: addr subc1 8 scan_in1 scan_out1 scan_en (inverted)
```

Example 2

The following example shows information written out by the report_scan_cells command following by the same information as it appears when it is written to the scan DEF file.

```
add_sub_chains wrapA schi1 scan_in1 scan_out1 1 mux_scan sen
add_subchain_clocks schi1 0 clk -first_cell_clock -leading
add_subchain_clocks schi1 0 clk -last_cell_clock -leading
add_sub_chains A schc1 scan_in2 scan_out2 1 mux_scan senout
add_subchain_clocks schc1 0 clk -first_cell_clock -leading
add_subchain_clocks schc1 0 clk -last_cell_clock -leading
add_sub_chains A scho1 scan_in3 scan_out3 1 mux_scan senout
add_subchain_clocks scho1 0 clk -first_cell_clock -leading
add_subchain_clocks scho1 0 clk -last_cell_clock -leading
add_sub_chains B schi2 scan_in1 scan_out1 1 mux_scan senin inverted
add_subchain_clocks schi2 0 clk2 -first_cell_clock -leading
add_subchain_clocks schi2 0 clk2 -last_cell_clock -leading
set_system_mode analysis
insert_test_logic
report_scan_cells
```

```
-------------------------------------------------------------------------------
          Chain     Group                                                  Clock
CellNo    Name             Name   Pathname      CellName  ScanOut   Clock  Polarity
-------------------------------------------------------------------------------
-         chain1           dummy  /lckup1          latch     Q      clk2   (-)
0         chain1 (schi2)   dummy  /uB/f3           sff       Q      clk2   (+)
1         chain1 (schi2)   dummy  /uB/f2           sff       Q      clk2   (+)
2         chain1 (schi2    dummy  /uB/f1           sff       Q      clk2   (+)
-         chain1           dummy  /lckup2          latch     Q      clk2   (+)
0         chain2 (schc1)   dummy  /uWA/uA/f21      sff       Q      clk    (+)
1         chain2           dummy  /ud              sff       Q      clk    (+)
2         chain2           dummy  /us              sff       Q      clk    (+)
0         chain3 (scho1)   dummy  /uWA/uA/f31      sff       Q      clk    (+)
-         chain4           dummy  /lckup3          latch     Q      clk    (-)
0         chain4 (schi1)   dummy  /uWA/uA/f3       sff       Q      clk    (+)
1         chain4 (schi1)   dummy  /uWA/uA/bb/f2    sff       Q      clk    (+)
2         chain4 (schi1)   dummy  /uWA/uA/f1       sff       Q      clk    (+)
-         chain4           dummy  /lckup4          latch     Q      clk    (+)
-------------------------------------------------------------------------------
```

Contents of the scan DEF file:

```
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1_sub0
  + START lckup2 Q
  + FLOATING
      uB/f1 ( IN SI ) ( OUT Q )
      uB/f2 ( IN SI ) ( OUT Q )
      uB/f3 ( IN SI ) ( OUT Q )
  + STOP lckup1 D
  # Partition for core chain in clock clk2 (pos-edge) domain
  + PARTITION partition_1  MAXBITS 3 ;

- chain2_sub0
```

```
  + START us Q
  + FLOATING
      ud ( IN SI ) ( OUT Q )
  + STOP uWA/uA/f21 SI
  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 1 ;

#  The following chain segment with only 1 or 2 scan cells has been
commented out for
#  compatibility with the layout tools.
#- chain3_sub0
#  + START uWA/uA/f31 SI
#  + STOP uWA/uA/f31 Q ;

- chain4_sub0
  + START lckup4 Q
  + FLOATING
      uWA/mux ( IN A1 ) ( OUT Y ) ( BITS 0 )
      uWA/uA/f1 ( IN SI ) ( OUT Q )
      uWA/uA/bb/f2 ( IN SI ) ( OUT Q )
      uWA/uA/f3 ( IN SI ) ( OUT Q )
  + STOP lckup3 D
  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 3 ;

END SCANCHAINS

END DESIGN
```

Related Commands

analyze_wrapper_cells
add_clocks
add_subchain_clocks
delete_sub_chains
report_scan_cells
report_sub_chains
set_wrapper_chains
write_scan_order
write_subchain_setup

add_subchain_clocks

Context: dft -scan, dft -test_points

Mode: setup

Usage

add_subchain_clocks ***subchain_name off_state*** {***clock_port_name***...}
[-Set | -Reset | -First_cell_clock | -Last_cell_clock]
[-LEading_edge | -Trailing_edge]

Description

Specifies the clock pins for scan chains within modules, library models, instances, blackboxes, or empty modules of a hierarchical design.

The specified clock pins are added relative to the module boundary, not the top-level of the design. The add_sub_chains command adds subchains to a specific instance or all instantiations of a specific module. When you use the add_subchain_clocks command to specify subchain clock pins on a module, the clock timing information is automatically adjusted for each instantiation of the module based on the connections to the top-level clock pins. The following information is considered to determine the clock timing for each subchain cell.

- Off-state value of the top-level clock pin (Usually specified via the add_clocks command)
- Overall inversions from the top-level clock pin to the clock pin of the subchain instance (Obtained through DRC rule checking of the input netlist)
- Off-state value of the subchain clock pin (Specified by the off_state argument of the add_subchain_clocks command)
- The clock capture edge of the sequential elements inside the subchain cell with respect to the off-state value of the subchain clock pin (Specified by the -leading_edge/-trailing_edge argument of the add_subchain_clocks command)

Note

The incorrect specification of the off_state argument and the -leading_edge/-trailing_edge switches may lead to incorrect placement of the subchain cells in chains, incorrect lockup cell insertion, or incorrect scan cell reporting. When specifying this information, consider all the inversions from the subchain clock pin down to the primitive sequential element inside the subchain instance in order to determine the correct clock capture edge. For further information, see the examples in the following sections.

Scannability analysis DRCs: S1, S2, and S3 validate the defined subchains when you exit Setup mode. Once you specify the subchain clock pins, you must also use set_test_logic to enable the insertion of the test logic to fix any S-rule violations.

Arguments

- ***subchain_name***

  A required string that specifies the name of a subchain previously defined with the add_sub_chains command.

- ***off_state***

  A required string that specifies the off state value for the subchain clock pin. Sequential instances maintain their state when the clock is at the specified value; either 1 or 0.

- ***clock_port_name***

  A required, repeatable string that specifies the clock pins on the subchain that must be controlled during scan chain shifting.

- -Set

  An optional switch that specifies the clock pins listed by the clock_port_name argument are *set* signals for cells in the subchain and should be held in the off state during scan chain shifting.

- -Reset

  An optional switch that specifies the clock pins listed by the clock_port_name argument are reset signals for cells in the subchain and should be held in the off state during scan chain shifting.

- -First_cell_clock

  An optional switch that specifies the clock pin listed by the clock_port_name argument drives the first cell in the subchain.

- -LAst_cell_clock

  An optional switch that specifies the clock pin listed by the clock_port_name argument drives the last cell in the subchain.

- -LEading_edge

  An optional switch that specifies the clock pin listed by the clock_port_name argument updates cells on the leading edge of the off-state to on-state transition. This value is only valid for defining the first and/or last cell clocks. This is the default value.

- -Trailing_edge

  An optional switch that specifies the clock pin listed by the clock_port_name argument updates cells on the trailing edge of the off-state to on-state transition. This value is only valid for defining the first and/or last cell clocks.

Example 1

The following example illustrates correct specification of the off_state argument along with -leading_edge/-trailing_edge switches. It also shows how clock capture edge is calculated by the tool.

The input netlist is given as below to insert scan chains to module TOP. Scan chains were inserted to modules A and B in previous sessions of the tool.

```
module TOP (tclk1, tclk2);
   input tclk1, tclk2;
      invx inv (.A(tclk1), .Y(tclk1_n));
      A    uA (.clk(tclk1_n));
      B    uB (.clk(tclk2));
   endmodule

   module A (clk, si, sen, so);
   input clk, si, sen;
   output so;
      invx inv (.A(clk), .Y(clk_n));
      sffn ff1 (.CK(clk_n), .SI(si), .SE(sen), .Q(net1));
      sffn ff2 (.CK(clk), .SI(net1), .SE(sen), .Q(net2));
      sffn ff3 (.CK(clk), .SI(net2), .SE(sen), .Q(so));
   endmodule

   module B (clk, si, sen, so);
   input clk, si, sen;
   output so;
      sffn ff1 (.CK(clk), .SI(si), .SE(sen), .Q(net1));
      sffn ff2 (.CK(clk), .SI(net1), .SE(sen), .Q(so));
   endmodule
```

The library model implementation for the scan flop 'sffn' is given below.

```
model sffn (D, SI, SE, CK, Q) (
   scan_definition (
      type = mux_scan;
         data_in = D;
      scan_in = SI;
      scan_enable = SE;
      scan_out = Q;
      non_scan_model = dffn (D, CK, Q);
   )
   input (D, SI, SE) ()
   input (CK) (negedge_clock)
   output(Q) (
         primitive = _inv (CK, CK_n);
         primitive = _mux (D, SI, SE, _D);
         primitive = _dff(, , CK_n, _D, Q, );
      )
   )
```

The top module shift clocks are defined as below.

add_clocks 0 tclk1
add_clocks 1 tclk2

The subchains on modules A and B are defined as below. The first flop of subchain 'subch1' (/uA/ff1), for example, is defined as trailing-edge with respect to the subhchain clock pin 'clk'. This edge value is calculated considering the off-state value of the subchain clock pin, the

inversion from the subchain clock pin to the subchain flop instance '/uA/ff1', and the inversion from the subchain flop instance clock pin to the _dff primitive inside the library model 'sffn'.

add_sub_chain A subch1 si so 3 mux_scan sen
add_subchain_clock subch1 1 clk -trailing_edge -first_cell_clock
add_subchain_clock subch1 1 clk -leading_edge -last_cell_clock
add_sub_chain B subch2 si so 2 mux_scan sen
add_subchain_clock subch2 0 clk -trailing_edge -first_cell_clock
add_subchain_clock subch2 0 clk -trailing_edge -last_cell_clock

After scan insertion for top module, the following is reported. Note that the cells of subchain 'subch2' are defined as trailing_edge, however, the tool adjusted the clock capture edge (ClockPolarity) as '+' based on the subchain's connection to the top module clock pin 'tclk2', with an off-state value of '1'. The clock capture edges for subchain 'subch1', remain the same with respect to the top module clock pin 'tclk1', since subchain clock pin is connected to the top module clock pin with an opposite off-state value through an inverter.

```
//  command: report_scan_cells
-------------------------------------------------------------------------
CellNo ChainName GroupName Pathname CellName ScanOut Clock ClockPolarity
-------------------------------------------------------------------------
   0 chain1 (subch1) grp1 /uA/ff3  sffn     Q       tclk1 (+)
   1 chain1 (subch1) grp1 /uA/ff2  sffn     Q       tclk1 (+)
   2 chain1 (subch1) grp1 /uA/ff1  sffn     Q       tclk1 (-)
   0 chain2 (subch2) grp1 /uB/ff2  sffn     Q       tclk2 (+)
   1 chain2 (subch2) grp1 /uB/ff1  sffn     Q       tclk2 (+)
-------------------------------------------------------------------------
```

Example 2

The following example defines a subchain on a library model and reports the subchain clocks. Note that all the clocks are defined.

add_sub_chains MULTIBITSFF3 chain1 SI SO 16 mux_scan S library_model
add_subchain_clocks chain1 0 RESET -reset
add_subchain_clocks chain1 0 SET -set
add_subchain_clocks chain1 0 Clk_c -first_cell_clock -trailing_edge
add_subchain_clocks chain1 0 Clk_x -leading_edge
add_subchain_clocks chain1 0 Clk_p -leading_edge
add_subchain_clocks chain1 0 Clk_a -last_cell_clock -leading_edge
report_subchain_clocks chain1

```
// clock name type             off_state edge
// ---------- ----             --------- -----
// RESET      reset            0
// SET        set              0
// Clk_c      first cell clock 0         trailing edge
// Clk_x                       0         leading edge
// Clk_p                       0         leading edge
// Clk_a      last cell clock  0         leading edge
```

Example 3

The following example reports the scannability problems of a subchain instance. The subchain is treated as a single sequential instance, but it is listed once for each clock line that needs fixing.

report_dft_check

```
SCANNABLE SUBCHAIN Testlogic /macro MULTIBITSFFX (1)
Set: T /macro/SET1
SCANNABLE SUBCHAIN Testlogic /macro MULTIBITSFFX (1)
Set: T /macro/SET2
SCANNABLE SUBCHAIN Testlogic /macro MULTIBITSFFX (1)
Set: T /macro/SET3
SCANNABLE SUBCHAIN Testlogic /macro MULTIBITSFFX (1)
Reset: T /macro/RESET1
```

All the test logic issues for a specified subchain instance map to a single DRC rule violation. The analyze_drc_violation command displays the entire macro (with an X on each clock line that requires test logic) and arbitrarily displays the driving gate for only one of the pins that requires fixing.

Related Commands

add_sub_chains
add_subchain_clocks
report_dft_check
report_subchain_clocks
write_subchain_setup

add_subchain_group

Context: dft -scan

Mode: setup

Usage

add_subchain_group ***object_name subchain_name … | subchain_name_expression …***
[-FLexible [-No_floating_cells] | -FIxed]

Description

Adds one subchain group to the system.

The add_subchain_group command specifies a group of subchains in a design so that the tool can stitch these subchains together. If you omit a subchain group, the insert_test_logic command chooses which subchains the tool stitches together. You must have added the subchains (using add_sub_chains) before you specify them as subchains in the argument list.

When adding the subchain groups, you can specify stitching constraints using the following options:

- **-FLexible (default)** — Permits unassigned single flip-flops and subchains in the scan chain for better balancing of top-level scan chains and allows breaking up the subchain group into multiple scan chains when necessary.
- **-FLexible [-No_floating_cells]** — Prevents the mixing of unassigned single flip-flops with subchains from flexible subchain groups in the same scan chain.
- **-Fixed** — Disallows *either* other flip-flops and subchains in the subchain group, or the breaking up of the subchain group. Stitches subchains in the order they are specified in the subchain_name or subchain_name_expression list.

Table 7-7 summarizes the types of subchains that can exist in a design.

Table 7-7. Subchain Types

Subchain Group	Multiple Chain Support	Stitching	Partitioning
No subchain group	Yes	Can be stitched with: • single cells • subchains that are not assigned to any subchain groups • subchains that are assigned to flexible subchain groups	-Clock/Edge merge -Number/Max_length
Flexible	Yes	Can be stitched with: • single cells • subchains that are not assigned to any subchain groups	-Clock/Edge merge -Number/Max_length

Table 7-7. Subchain Types

Subchain Group	Multiple Chain Support	Stitching	Partitioning
Flexible -No_floating_cells	Yes	Can be stitched with: • no other cells or subchains	-Clock/Edge merge -Number/Max_length
Fixed	No	Can be stitched with: • no other cells or subchains	A new scan chain is added per add_subchain_group command

Arguments

- ***object_name***

  A required string specifying the subchain group name.

- ***subchain_name***

  A required repeatable string identifying the subchain name for inclusion into the subchain group. Before identifying the subchain, you must add it with add_sub_chains. You can also report subchain information using report_sub_chains.

  If you added a subchain to a subchain group on a module-based basis, and the module is instantiated multiple times, the tool includes the subchains on all instances of the module in the subchain group when this subchain is specified in the list of subchains of the add_subchain_group command. If the subchains on different instances of the module are to be put into different subchain groups, these subchains need to be added instance-based instead of module-based in order to be assigned a unique subchain name. For more information about adding subchains as module-based versus instance-based, see add_sub_chains.

- ***subchain_name_expression …***

  A required repeatable string in regular expression form that represents a list of subchain names. The regular expression may include asterisk (*), or question mark (?), or wildcard characters.

Note

If a wildcard character is part of a subchain name, you must use the escape character, (\), at the beginning of the name when specifying it as a subchain_name argument for this command. Additionally, you must surround the entire subchain name with curly brackets in order to bypass the Tcl command interpreter. For example, the subchain name sub*?123 should be specified for the subchain_name argument as {sub*\?123}.

The tool adds all subchains whose names match this regular expression to the subchain group specified by the object_name argument.

- **-FLexible** — An optional switch that specifies the type of the subchain group. This is the default. With this option, the tool permits unassigned single flip-flops and subchains in the scan chain for better balancing of the top-level scan chains. Using this option also permits breaking up the subchain group into multiple scan chains when necessary. The tool partitions the subchains of flexible subchain groups along with the scannable flip-flops of the design according to your insert_test_logic command's -Clock/Edge merge and -Number/Max_length options.
- **-FLexible [-No_floating_cells]** — The -No_floating_cells option prevents the mixing of unassigned single flip-flops with subchains from flexible subchain groups in the same scan chain.
- **-Fixed** — In contrast, the -Fixed option disallows either other flip-flops and subchains in the subchain group, or the breaking up of the subchain group. With this option, the subchains of a fixed-type subchain group are placed in a single chain and are not partitioned according to insert_test_logic command options. No other flip-flops or subchains are allowed in the scan chain of a fixed-type group. Note that the subchains of a fixed-type subchain group are stitched into a single scan chain in the order they are specified in the subchain_name or subchain_name_expression list.

Examples

The following example defines a pre-existing scan subchain:

```
add sub chain subblockA subchain1 /scan_in1 /scan_out1 250 \
                    mux_scan /scan_en -module
add sub chain subblockC subchain2 /scan_in1 /scan_out1 120 \
                    mux_scan /scan_en -instance ...
report_sub_chains
```

```
mux_scan: subblockA subchain1 scan_in1 scan_out1 scan_en
mux_scan: subblockC subchain2 scan_in1 scan_out1 scan_en
```

```
add_subchain_group subchaingroup1 subchain1 subchain2 -fixed
```

Related Commands

add_sub_chains
add_subchain_group
delete_sub_chains
delete_subchain_groups
insert_test_logic
report_subchain_groups
report_sub_chains

add_tied_signals

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

Mode: setup

Usage

add_tied_signals {**0** | **1** | **X** | **Z**} ***floating_object_name***... [-Pin]

Description

Adds a value to floating signals or pins.

The add_tied_signals command assigns a specific value to not-clearly-defined floating signals or pins. If there are floating signals or pins in the design, a warning appears when you leave the setup mode. If you do not assign a specific value, the tool ties the signal or pin values to the default value. To change the default tied value, use the set_tied_signals command.

When you add_tied_signals or pins, the tool places them into the user class. This includes instance-based, blackbox tied signals. When the netlist ties signals or pins to a value, the tool places them into the system class.

Note

The tool will not tie a signal that is connected to bidirectional pins. For example, with a Verilog netlist that has a top level pin named Vdd of type “inout”, you cannot use “add_tied_signals 1 Vdd -pin” to tie Vdd high.

Arguments

- **0**

  A literal that ties the floating nets or pins to logic 0 (low to ground).

- **1**

  A literal that ties the floating nets or pins to logic 1 (high to voltage source).

- **X**

  A literal that ties the floating nets or pins to unknown.

- **Z**

  A literal that ties the floating nets or pins to high-impedance.

- ***floating_object_name***

  A required, repeatable string that specifies the floating nets or pins to which you want to assign a specific value. The tool assigns the tied value to all floating nets or pins in all modules that have the names you specify.

  For pin names, you must include the -Pin switch on the command line; otherwise, the tool assumes the name is a net name.

- -Pin

  An optional switch specifying that the *floating_object_name* argument that you provide is a floating pin name.

Example

The following example ties all floating signals in the circuit that have the net names vcc and vdd, to logic 1 (tied to high):

add_tied_signals 1 vcc vdd

Related Commands

add_black_boxes
delete_tied_signals
report_tied_signals
set_tied_signals

add_to_collection

Context: unspecified, all contexts

Mode: all modes

Usage

add_to_collection ***base_collection object_spec*** [-unique]

Description

Adds objects to a base collection and returns the new collection without affecting the original base collection.

By default, objects that exist in both the base_collection and the object_spec are duplicated in the resulting collection. You can use the -unique switch to remove duplicates.

Arguments

- ***base_collection***

  A required value that specifies the base collection to which objects are to be added. This base collection is copied to a new collection, and objects matching object_spec are added to the new collection. The base_collection can be an empty collection (empty string).

- ***object_spec***

  A required value that specifies an object name, a Tcl list of object names, or a collection of objects to add to the base_collection. The object_spec can be an empty collection (empty string). If object_spec is empty, the result is a copy of the base collection.

- -unique

  An optional switch that removes duplicate objects from the resulting collection. By default, duplicate objects are not removed.

Example 1

The following example adds two collections, a and b, to create a new collection called sum.

```
set a [get_ports a*]
set b [get_ports b*]
set sum [add_to_collection $a $b]
```

Example 2

The following example creates a collection of all nets connected to pins of design instances matching u1/myPrefix*. Note that using the -unique option automatically removes duplicate nets.

```
set sum “”
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
        set inputs [get_pins -of_instance $inst -direction input]
        set outputs [get_pins -of_instance $inst -direction output]
        set sum [add_to_collection $sum [get_fanins $inputs -stop_on net] -unique]
        set sum [add_to_collection $sum [get_fanouts $outputs -stop_on net] -unique]
}
```

Note

This example could have used the append_to_collection command instead, which would have been more efficient because in this example, the intermediate collections are destroyed as soon as they are added.

Related Commands

append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

add_write_controls

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

Mode: setup

Usage

add_write_controls {**0** | **1**} ***primary_input_pin***...

Description

Specifies the off-state value of the write control lines for RAMs.

The add_write_controls command defines the circuit write control lines and assigns their off-state values. The off-state value of the pins you specify must be sufficient to keep the RAM contents stable.

Note: The tool automatically adds a write control to the clock list during flattening if the flattening process reaches the clock pin of an edge-triggered RAM. In this case, you must ensure that your timeplate includes the waveform for the newly defined clock.

Arguments

- **0**

  A literal specifying 0 is the off-state value for the RAM primary_input_pin.

- **1**

  A literal specifying 1 is the off-state value for the RAM primary_input_pin.

- ***primary_input_pin***

  A required, repeatable string specifying the primary input pins that are write control lines for the RAM and to which you want to assign an off-state value.

Example

The following example assigns an off-state to two write control lines, w1 and w2:

```
add_write_controls 0 w1 w2
set_system_mode analysis
create_patterns
```

Related Commands

analyze_control_signals
delete_write_controls
report_write_controls

analyze_atpg_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Usage

analyze_atpg_constraints {-AUto | -ALl | [{pin_pathname | gate_id# | function_name}... [-Satisfy | -Exclusive]]} [-Bus]

Description

Checks the ATPG constraints you created for their satisfiability or mutual exclusivity.

If you issue the analyze_atpg_constraints command without any arguments, the default is -All. When the command finishes, the tool displays a message that indicates whether the analysis passed, failed, or aborted the ATPG constraint analysis.

Note

With EDT on, Tessent TestKompress support of this command is limited: it will not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

Arguments

The following lists the three methods for naming the objects for which you want to analyze the constraints. You can use any number of the three argument choices, in any order.

If you only specify an object name when you issue this command, by default, the tool performs the satisfiability (-Satisfy) analysis.

- -Auto

  An optional switch that automatically tries to locate the atpg constraint that cannot be satisfied. The analysis checks to see if any single constraint cannot be satisfied, then reports each constraint that cannot be satisfied (given the current abort limit and other restrictions). Sometimes, each constraint can be satisfied by itself, but some set of constraints cannot all be satisfied. In this case, -Auto switch proceeds to a second analysis where it adds atpg constraints to a set to create a minimal set that can’t be satisfied.

- -ALl

  An optional switch that specifies for the tools to perform the ATPG analysis simultaneously for all the current ATPG constraints. If you do not specify an object name, this is the command default.

- pin_pathname

  A repeatable string that specifies the pathname to the pin on which you are analyzing the constraints.

- gate_id#

  A repeatable integer that specifies the gate identification number of the gate on which you want to analyze the constraints.

- function_name

  A repeatable string that specifies the name of a function you created with the add_atpg_functions command. If you generated the ATPG function with the -Cell option and added constraints with the -Cell option, then the tool also analyzes the constraints on all the cells affected by that ATPG function.

- -Satisfy

  An optional switch that specifies for the ATPG process to attempt to create a pattern that satisfies the selected ATPG constraint. This is the default if you specify an object name without a switch. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the -Bus switch).

  When the command finishes, the tools display a message that indicates whether the analysis passed (the ATPG process successfully generated a pattern), failed (the ATPG process could not find any possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis passes, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- -Exclusive

  An optional switch that specifies for the ATPG process to attempt to create a pattern that sets the selected ATPG constraint at a value different from its constrained value. This test's intent is to ensure that such a pattern does not exist. During the ATPG process, the test generator does not consider the effect of other ATPG constraints or bus contention prevention (unless you use the -Bus switch).

  When the command finishes, the tool displays a message that indicates whether the analysis passed (the ATPG process could not find any possible pattern), failed (the ATPG process found another possible pattern), or aborted (the ATPG process gave up on trying to find a successful pattern). If the analysis fails, the data that the tool simulated for the pattern is available in parallel pattern zero (0).

- -Bus

  An optional switch that specifies for the tool to consider bus contention prevention during the ATPG process.

Example

The following example creates an ATPG constraint and then checks for mutual exclusivity:

add_atpg_constraints 1 435
analyze_atpg_constraints 435 -exclusive

```
// ATPG constraint 435=1 failed mutual exclusivity check (data
                                       in parallel pattern 0).
```

Related Commands

add_atpg_constraints

analyze_bus

Context: dft -edt, patterns -scan

Mode: analysis

Usage

analyze_bus {***gate_id#***... [-Exclusivity | -Prevention | -Zstate]} |
-Drc_check | -ALl | -Auto | -ANAlyze_sequentially

Description

Analyzes the specified bus gates for contention problems.

If the bus passes the analysis, the tool displays a message indicating that it did so. If the analysis aborts, the tool displays a message that identifies the tri-state drivers (TSDs) the tool was analyzing at abort time. If the bus fails the analysis, the tool displays a message that identifies the two offending TSDs (the tri-state drivers capable of being on simultaneously, while driving different values).

Note

After using the analyze_bus command, the set_gate_report Parallel_pattern 0 command may display conflicting values if failures are found. That is, it is possible to have TIE-X gates with a 1/0 as an output. This is due to the analyze_bus command setting values to determine contention states. This analysis continues until a conflict is found.

In the tool, analyze_bus only shows the values that were used by the ATPG, and which the command considers relevant to the bus conflict. As the purpose of the analysis is to determine TSD input combinations that result in bus conflicts, the command does not simulate the output values of the TSD.

Note

With EDT on, Tessent TestKompress support of this command is limited: it will not tell you if a constraint cannot be satisfied with EDT compression. It may also falsely report a constraint can be satisfied.

The commands “set_contention_check On -Atpg” and “set_iddq_checks -Bus” both cause ATPG to ensure that every bus is contention free during deterministic test generation. Sometimes, this requirement cannot be met for many or all of the faults targeted by ATPG, preventing you from obtaining adequate fault coverage. When this happens, the analyze_bus command determines which bus or buses cannot be made contention free, so that you can investigate the circuit around this bus to find out what is preventing contention- free tests.

When you issue this command, you must either specify a *gate_id#* value or one of the global switches (-Drc_check or -All).

Arguments

- ***gate_id#*** -Exclusivity | -Prevention | -Zstate

  A repeatable integer with an optional switch that specifies the identification number of the bus gate and the type of analysis you want the tool to perform.

  The available switch choices are as follows:

  -Exclusivity — An optional switch that specifies for the tool to analyze the bus gate to see if it has mutual exclusivity. Mutual exclusivity means that only one driver can simultaneously force a strong signal onto the bus. Exclusivity is the default behavior when you specify a *gate_id#* value without a corresponding switch.

  -Prevention — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a state of non-contention.

  -Zstate — An optional switch that specifies for the tool to analyze the bus gate for its ability to attain a high-impedance (Z) state.

- **-Drc_check**

  A switch that specifies for the tool to run the design rule check process again to categorize all buses and display the results. This is useful if you are changing constraints or the abort limit in an attempt to pass bus checks rather than abort.

- **-ALl**

  A switch that specifies for the tool to use the more extensive ATPG process to place all the fail and abort buses in a noncontentious state. The internal simulation data for this pattern is available in parallel pattern 0.

- **-AUto**

  A switch that automatically tries to locate the bus that cannot be made contention free. The analysis checks to see if any single bus cannot be made contention free. Each bus that cannot be made contention free (given the current abort limit and other restrictions) is reported to the user. Sometimes, each bus can be satisfied by itself, but some set of buses cannot all be satisfied. In this case, -Auto switch proceeds to a second analysis where it creates a minimized set of buses that can't be satisfied. The final, reduced set of buses, which cannot all be made contention free, is reported.

- **-ANAlyze_sequentially**

  A switch that specifies to perform a sequential analysis to find bus contention problems. For each clock defined using the add_clocks command, the sequential analysis applies a 1-cycle test, then a 2-cycle test, and so on, up to the reporting depths for which the tests fail. It then analyzes and lists the specific buses causing the failures at those depths.

Note

The same sequential analysis is performed automatically by the analyze_restrictions command if that command finds bus contention is the problem preventing success.

Example 1

The following example analyzes a bus that failed the regular bus contention checking:

set_system_mode analysis
analyze_bus 493

Example 2

The following example displays the current categorization of bus gates, and then performs the prevention check on a specific bus gate:

set_system_mode analysis
analyze_bus -drc_check

```
// ATPG bus checking results: pass=1, bidi=1, fail=0, abort=0,
                                        CPU time=0.00.
```

analyze_bus 495 -prevention

```
// Controllability justification was successful (data
                     accessible using parallel_pattern 0).
// Pattern type: Basic_scan
```

Example 3

The following example analyzes a bus that failed the regular bus contention checking. Notice the input values of gates 7 & 8 (the two TSDs) show that ATPG has correctly illustrated a conflict condition:

```
//  Warning: BUS gate /Y (9) has possible contention
      on drivers 7 and 8. (E10-1)
```

analyze_bus 9

```
//  ATPG bus checking performed on /Y (9) for 2 bused TSDs.
//  Failure occurred while checking TSDs 7 and 8
      (data in parallel_pattern 0).
```

set_gate_report parallel 0
report_gates 9

```
//  /Y (9)  BUS
//      "I0"   I  (X)  7-/tri01/Y
//      "I1"   I  (X)  8-/tri02/Y
//      "OUT"  O  (X)  10-/Y
```

report_gates 7

```
//  tri01 (7)  TSD
//      E      I  (1)  3-/E
//      "D"    I  (1)  5-
//      Y      O  (X)  9-
```

report_gates 8

```
//  tri02 (8)  TSD
//      E      I  (1)  4-/E1
//      "D"    I  (0)  6-
//      Y      O  (X)  9-
```

Related Commands

analyze_restrictions
report_bus_data
set_contention_check
set_gate_level
set_gate_report

analyze_compression

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Determine maximum possible compression (maximum chain:channel ratio)

analyze_compression [-Preserve_faults_from_analysis]
[-OVerride_user_settings | -NO_AUto]
[-COMpaction_effort {Low | High | Maximum | *integer*}] [-NO_Analyze]
[-COVerage_effort {Low | High}]
[-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg]
[-PATterns_per_pass integer]

Emulating a virtual single block EDT configuration

analyze_compression [-Preserve_faults_from_analysis]
[-COMPACTOR_type Xpress | Basic]
[-SHift_power_control [-SWitching_threshold_percentage *int*]]
[-SEParate_control_data_channels]
[-CHAINs *N*] {[-CHANNELs *M*] | [{-INPut_channels *I*}{-OUtput_channels *O*}]}
[-OVerride_user_settings | -NO_AUto]
[-COMpaction_effort {Low | High | Maximum | *integer*}] [-NO_Analyze]
[-COVerage_effort {Low | High}]
[-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg]
[-PATterns_per_pass *integer*]

Emulating a virtual modular EDT configuration

analyze_compression [-Preserve_faults_from_analysis]
[-COMPACTOR_type Xpress | Basic]
[-SHift_power_control [-SWitching_threshold_percentage *int*]]
[-SEParate_control_data_channels]
{-Edt_block *block1_name* [-CHAINs *int*] { [-CHANNELs *int*]
[-INPut_channels *int* -OUtput_channels *int*]}}
[{-Edt_block *block2_name* [-CHAINs *int*] { [-CHANNELs *int*] |
[-INPut_channels *int* -OUtput_channels *int*]}} ...]
{[-SHARE_data_channels [{*block3_name block4_name* ...}]
[-DATA_and_control_channels [*int*]]]} ...
[-OVerride_user_settings | -NO_AUto]
[-COMpaction_effort {Low | High | Maximum | integer}] [-NO_Analyze]
[-COVerage_effort { Low | High }]
[-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg]
[-PATterns_per_pass integer]

Description

Analyzes the compression for an application in two steps.

The two steps are as follows:

1. Analyzes a scan-inserted design and returns a range of maximum chain:channel ratios where the test coverage begins to decline.
2. Emulates a hardware configuration for a specified chain:channel ratio, generates temporary test patterns, and returns test data statistics for the compression configuration.

The analyze_compression command uses the fault model type specified by the set_fault_type command. If no fault model type is specified, the tool uses the "stuck" fault model by default. The analyze_compression command uses the current fault population. If no faults are added, the tool operates on all faults or a subset of sampled faults that are determined by the fault sampling rate specified by the "set_fault_sampling <rate>" command.

To enable N-detection during analysis, you must first issue the set_multiple_detection command. After enabling N-detection, you can run the analyze_compression command with the -Preserve_faults_from_analysis switch and the tool preserves the N-detection information for all the detected faults; if you do not specify the -Preserve_faults_from_analysis switch, the tool restores the N-detection information for all detected faults to the states prior to the execution of the analyze_compression command. For an example of enabling N-detection, see Example 16.

Note

During emulation, if the emulated EDT block has the same name as the original one, the tool preserves all the scan cells in the original EDT block; if you specify new EDT block names, the tool tries to allocate all scan cells into the new EDT blocks to get balanced scan chains. You cannot mix the names of the original EDT blocks with the names of the newly emulated EDT blocks. This is illustrated in Example 17.

For more information, see "Analyzing Compression" in the *Tessent TestKompress User's Manual*.

Note

When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, TestKompress assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

Arguments

- -Preserve_faults_from_analysis

  An optional switch that preserves the fault status at the end of the analyze_compression operation based on the APTG run of the command even though no patterns are stored. By default, without the -Preserve_faults_from_analysis switch, the fault status at the end of the analyze_compression operation is returned to its status prior to the execution of the command. This switch can be used to experiment with analyzing faults remaining after an analysis run.

  If the -Preserve_faults_from_analysis switch is specified, the tool preserves the N-detection information for all the detected faults during the analyze_compression operation.

Note

When you specify this option, if there are already internal patterns generated by a previous create_patterns command, the tool deletes all internal patterns and issues a warning stating that the previously generated patterns are deleted because they will be inconsistent with the fault population.

- -OVerride_user_settings

  For information about this switch, see the create_patterns command.

- -NO_AUto

  For information about this switch, see the create_patterns command.

- -Compaction_effort {Low | High | Maximum | integer}

  For information about this switch, see the create_patterns command.

- -NO_Analyze

  For information about this switch, see the create_patterns command.

- -COVerage_effort { Low | High }

  For information about this switch, see the create_patterns command.

- -NO_REDundancy_analysis

  For information about this switch, see the create_patterns command.

- -NO_TERMINATE_ineffective_atpg

  For information about this switch, see the create_patterns command.

- -PATterns_per_pass integer

  For information about this switch, see the create_patterns command.

- -COMPACTOR_type Xpress | Basic

  An optional switch that specifies the compactor to use for the analysis. By default, the Xpress compactor is used. The specified compactor is applied to all blocks in a block-level

application. This switch cannot be specified more than once; otherwise, an error message is generated.

- -SHift_power_control [-SWitching_threshold_percentage *int*]

  An optional switch and switch-integer pair that specifies low-power usage for the analysis. You must specify the -Channel option when using this -Shift_power_control option.

  If low-power usage is specified, the tool uses the specified low-power controller for all blocks. If no low-power usage is specified, then no low-power is used. This switch cannot be specified more than once; otherwise, an error message is generated.

  If you do not specify the -switching_threshold_percentage option with the -shift_power_control switch, the default is set to 15%.

  The specified low-power usage applies to all blocks in a block-level application.

- -SEParate_control_data_channels

  An optional switch that, if used, tells the tool to separate control and data channels for all EDT blocks during emulation, regardless of whether channel sharing is used. If this switch is not used, the tool does not separate control and data channels for all EDT blocks unless the block is in a channel sharing group specified with the -SHARE_data_channels switch.

- {-Edt_block *block1_name*} ...

  An optional, repeatable, switch and string pair that specifies the name of a design block containing compression logic (compression block). This option specifies multiple compression blocks for the analysis of a block-level architecture design.

  If you only specify one compression block, the design is treated as a chip-level architecture for analysis.

  Chain and/or channel specifications must be defined for each block individually.

  If you do not specify any compression blocks for a block-level design, the design is analyzed with the configurations defined in the design.

  Only the associated scan cells are used to analyze existing compression blocks. If you do not specify any scan channels for existing blocks, the tool uses those specified in the design.

  If you specify any new compression blocks, the tool completely ignores the scan chain and channel configurations defined in the design. For the analysis, the tool distributes the scan cells across all specified blocks in a way that balances the scan chain length for all blocks. In this case, you must specify both chains and channels for each block. You cannot define both new and existing compression blocks in one design.

- -CHAINs *int*

  An optional switch and integer pair that specifies how many scan chains to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. If no scan chains are specified, the scan chains defined in the design are used.

- [-CHANNELs *int*] | [{-INPut_channels *int*}{-OUtput_channels*int*}]

  An optional switch and integer pair or set of switch and integer pairs that specifies the channels to use when emulating a chain:channel ratio for test data volume and fault coverage analysis. Channels must be specified when analyzing a design without compression logic inserted.

  If no channels are specified for a design with compression logic inserted, the following scenarios apply:

  - **Block-level architecture** — The sum of the channels defined for each block is used, and the design is emulated as a single compression block.
  - **Chip-level architecture** — The channels defined by the existing compression logic are used.

  Use the -input_channels/-output channels arguments to specify an asymmetrical channel configuration. These arguments must be used in pairs.

- {[-SHARE_data_channels [{*block3_name block4_name* ...}] [-DATA_and_control_channels [*int*]]]} ...

  An optional argument used to construct one or more channel sharing groups such that the EDT blocks in each channel sharing group share data input channels with each block having individual control channels such that the total number of input channels used in this group equals the input channel number specified by -DATA_and_control_channels. The tool calculates the minimum number of input channels based on the channel sharing groups you define with this argument.

The block_name parameters are the names of the blocks defined to be in the channel sharing group.

The following rules apply to the creation of channel sharing groups:

- If no block_name parameters are specified, the design must already have two or more EDT blocks; otherwise, the tool generates an error.
- If no block_name parameters are specified, all blocks share the data channels; in this case, only one channel sharing group is allowed.
- If block_name parameters are specified, each channel sharing group must have a minimum of two blocks defined; otherwise, the tool generates an error.
- An EDT block can only be defined in one channel sharing group. Note, an EDT block does not have to be defined in any channel sharing group.
- If a block is defined in a channel sharing group, the set_edt_options "-separate_control_data on" switch is automatically set. In this case, the tool calculates the number of control channels that are needed for the block by determining the number of control bits and the length of data channels. The total input channel count for each block, specified by either the analyze_compression "-channels" or "-input_channels"

switches, must be greater than or equal to the number of control channels plus 1. This means the block must have at least one data channel.

- The specified input pin count (N) for each channel sharing group must be greater than or equal to the sum of the #control channels on each block in the group plus 1.That is, at least one pin must be shared by the data channels of the EDT blocks in one channel sharing group.
- The number of input data pins available to be shared in the group cannot be greater than or equal to the sum of all data channels of all blocks in the group.
- The number of input data pins available to be shared in this group must be greater than or equal to the maximum number of data input channels of all EDT blocks in the group.

Example 1

The following example analyzes a design and returns the upper range of chain:channel ratios:

analyze_compression

```
// For stuck-at faults
//
//  Chain:Channel Ratio  Predicted Fault Coverage Drop
//  -------------------  ------------------------------
//  153                  negligible fault coverage drop
//  154                  0.01 % - 0.05 % drop
//  160                  0.10 %
//  168                  0.15 %
//  171                  0.20 %
…
//  CPU time is 155 seconds.
```

Example 2

The following example emulates compression logic with 800 scan chains and 8 scan channels, generates test patterns, and displays test pattern statistics for a chip-level architecture:

set_fault_sampling 1
analyze_compression -chains 800 -channels 8

```
            Statistics Report
              Stuck-at Faults
-------------------------------------------
Fault Classes                     #faults
                                  (total)
---------------------------  --------------
  FU (full)                       59164
  -------------------------  --------------
  UO (unobserved)               167 ( 0.28%)
  DS (det_simulation)         47815 (80.82%)
  DI (det_implication)        10425 (17.62%)
  UU (unused)                   123 ( 0.21%)
  TI (tied)                     219 ( 0.37%)
  BL (blocked)                   50 ( 0.08%)
  RE (redundant)                142 ( 0.24%)
  AU (atpg_untestable)          223 ( 0.38%)
-------------------------------------------
```

```
Untested Faults
  -------------------------
  AU (atpg_untestable)
    SEQ  (sequential_depth)     20 ( 0.03%)
    Unclassified               203 ( 0.34%)
  UC+UO
    AAB  (atpg_abort)            8 ( 0.01%)
    UNS  (unsuccess)           159 ( 0.27%)
-------------------------------------------
Coverage
  -------------------------
  test_coverage                     99.33%
  fault_coverage                    98.44%
  atpg_effectiveness                99.72%
-------------------------------------------
#test_patterns                         348
#simulated_patterns                    416
CPU_time (secs)                      250.6
-------------------------------------------

Note: The reported statistics are based on a 1% fault sample.

//  CPU time to analyze compression is 213 seconds.
//
//  -----------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 8
//     shift cycles : 164+4
//  -----------------------------------------------------
//  pattern       # test  # scan                   volume
//  type        patterns   loads  (cell loads or unloads)
//  ----------  --------  ------  -----------------------
//  chain_test       130     130                   174720
//  basic            348     348                   467712
//  ----------  --------  ------  -----------------------
//  total            478     478                   642432   (642.4K)
//
```

Example 3

The following example emulates the specified compression logic, generates test patterns, and displays test pattern statistics for a block-level architecture:

set_fault_sampling 5
analyze_compression -edt_block edt_bk1 -chains 400 -channels 2
-edt_block edt_bk2 -chains 2000 -channels 10

```
            Statistics report
-------------------------------------------
                           #faults  #faults
fault class                (coll.)  (total)
-----------------------    -------  -------
FU (full)                   21712    31930
-----------------------    -------  -------
UO (unobserved)                14       26
DS (det_simulation)         17203    27194
DI (det_implication)         4239     4374
```

```
      PU (posdet_untestable)          6       7
      UU (unused)                    29      51
      TI (tied)                      22      22
      RE (redundant)                112     116
      AU (atpg_untestable)           87     140
      -----------------------   -------   -------
      test_coverage               99.52%  99.47%
      fault_coverage              98.77%  98.88%
      atpg_effectiveness          99.94%  99.92%
      ---------------------------------------------
      #test_patterns                          700
      #simulated_patterns                     704
Note: The reported statistics are based on a 5% fault sample.

CPU time to analyze compression is 70 seconds.

//  ------------------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 1
//     shift cycles : 487+49
//  ------------------------------------------------------------
//  pattern             # test  # scan                   volume
//  type              patterns   loads  (cell loads or unloads)
//  ----------------  --------  ------  -----------------------
//  chain_test             265     265                   142040
//  basic                 6426    6426                  3444336
//  clock_po                10      10                     5360
//  clock_sequential      1763    1763                   944968
//  ----------------  --------  ------  -----------------------
//  total                 8464    8464                  4536704   (4.5M)
//
```

Example 4

In the following example, a fault model type is not specified. By default, the stuck-at fault model type is used.

analyze_compression -chain 100 -channel 2

```
// ************************************
// * Run ATPG for stuck-at faults.
// ************************************
```

Example 5

In the following example, the -Preserve_faults_from_analysis switch is used to update the fault detection status based on the ATPG results of the analyze_compression command.

analyze_compression -chain 100 -channel 2 -Preserve_faults_from_analysis

```
// ************************************
// * Run ATPG for stuck-at faults.
// ************************************

Note: Fault detection status will be updated based on the ATPG results of
analyze_compression even though no patterns are stored.
```

Example 6

In the following example, the set_fault_type command is used to specify the fault model type to be used by the analyze_compression command.

```
set_fault_type transition
analyze_compression -chain 100 -channel 2
```

```
// ************************************
// * Run ATPG for transition faults.
// ************************************
```

Example 7

In the following example, the set_fault_type command is used to specify that a user-defined fault model type is to be used by the analyze_compression command.

```
set_fault _type UDFM
read_fault_sites ../fault/sampled.udfm
analyze_compression -chain 100 -channel 2
```

```
// ************************************
// * Run ATPG for UDFM faults.
// ************************************
```

Example 8

In the following example, the Xpress compactor type is specified for a single EDT block.

```
analyze_compression -compactor_type xpress -chain 100 -channel 2
```

Example 9

In the following example, both EDT blocks are emulated with the Xpress compactor.

```
analyze_compression -compactor_type xpress \
        -edt_block blk1 -chain 100 -channel 2 \
        -edt_block blk2 -chain 200 -channel 2
```

Example 10

In the following example, both EDT blocks are emulated with the Basic compactor.

```
analyze_compression
        -edt_block blk1 -compactor_type basic -chain 100 -channel 2 \
        -edt_block blk2 -chain 200 -channel 2
```

Example 11

In the following example, only one EDT block is emulated with the low-power controller.

```
analyze_compression -shift_power_control -switching_threshold_percentage 20 \
        -chain 100 -channel 2
```

Example 12

In the following example, the low-power controller is used for both EDT blocks. The switching_threshold_percentage switch is set to 20%.

```
analyze_compression -shift_power_control -switching_threshold_percentage 20 \
        -edt_block blk1 -chain 100 -channel 2 \
        -edt_block blk2 -chain 200 -channel 2
```

Example 13

In the following example, both EDT blocks are emulated with the same low-power controller.

```
analyze_compression -edt_block blk1
        -shift_power_control -switching_threshold_percentage 20 -chain 100 -channel 2 \
        -edt_block blk2 -chain 200 -channel 2
```

Example 14

In the following example, only one EDT block is emulated with the low-power controller. The switching_threshold_percentage switch is set to 15% by default.

```
analyze_compression -shift_power_control -chain 100 -channel 2
```

Example 15

In the following example, both EDT blocks are emulated with the low-power controller. The switching_threshold_percentage switch is set to 15% by default.

```
analyze_compression -shift_power_control -edt_block blk1 -chain 100 -channel 2 \
        -edt_block blk2 -chain 200 -channel 2
```

Example 16

The following example enables N-detection ATPG during the analyze_compression.

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1
set_fault_sampling 2
analyze_compression -chain 100 -channel 2
```

```
//  ************************************
//  *   Run ATPG for stuck-at faults.
//  ************************************
……

  ----------------------
  test_coverage                                      78.79%
  fault_coverage                                     76.37%
  atpg_effectiveness                                 79.44%
-------------------------------------------------------------
#test_patterns                                          617
#simulated_patterns                                    1280
CPU_time (secs)                                        61.0
```

```
-------------------------------------------------------------
Multiple Detection Statistics
-------------------------------------------------------------
  Detections                      DS Faults    Test Coverage
     (N)                   (Detection == N) (Detection >= N)
------------------------  ---------------- ----------------
      1                        4 (  0.29%)   1070 ( 78.79%)
      2                        9 (  0.64%)   1066 ( 78.50%)
      3                      206 ( 14.70%)   1057 ( 77.84%)
      4                       27 (  1.93%)    851 ( 62.67%)
      5+                     727 ( 51.89%)    824 ( 60.68%)
-------------------------------------------------------------
bridge_coverage_estimate                            74.79%
-------------------------------------------------------------

Note: The reported statistics are based on a 2% fault sample.

//  CPU time to analyze compression is 30 seconds.
//
//  -----------------------------------------------------
//  Scan volume report.
//  -------------------
//     channels     : 4
//     shift cycles : 41+19
//  -----------------------------------------------------
//  pattern       # test  # scan                  volume
//  type        patterns   loads  (cell loads or unloads)
//  ----------  --------  ------  -----------------------
//  chain_test        36     136                    32640
//  basic            617     617                   148080
//  ----------  --------  ------  -----------------------
//  total            753     753                   180720   180.7K)
```

Example 17

The following examples demonstrate how the tool enforces the naming requirements for the emulation step.

Case 1

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block2 -chain10 -channel 1
```

The tool maintains all cells in the original block1 and block2 because they have the same name as the original, but changes the number of scan chains and channels based on the command.

Case 2

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1 \
                    -edt_block block4 -chain10 -channel 1
```

The tool does not maintain all of the cells in the original blocks because the names changed. The tool reallocates cells to make all of the chains in the two blocks as balanced as possible.

Case 3

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block4 -chain10 -channel 1
```

The tool does not allow the mixing of the original names and the newly specified names and generates an error message.

Case 4

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block1 -chain10 -channel 1
```

The tool does not allow this case because it changes two blocks to one block, but still uses the same name as before.

Case 5

The original design has two EDT blocks named block1 and block2.

```
analyze_compression -edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Case 6

The original design has one EDT block named block1.

```
analyze_compression -edt_block block1 -chain10 -channel 1 \
                    -edt_block block2 -chain10 -channel 1
```

The tool does not allow this and displays an error message.

Case 7

The original design has one EDT block named block1.

```
analyze_compression -edt_block block2 -chain10 -channel 1 \
                    -edt_block block3 -chain10 -channel 1
```

The tool supports this case.

Example 18

The following examples demonstrate how the tool supports channel sharing.

Case 1:

The design has one sharing group with 12 input channels shared between all blocks (block1, block2, block3 and block4).

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -share_data_channels -data_and_control_channels 12
```

Case 2:

The design has two sharing groups. The first group has 6 input channels shared between block1 and block2. The second group has 8 input channels shared between block3 and block4.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -edt_block block4 -chain 200 -channel 2 \
                    -share_data_channels block1 block2 -data_and_control_channels 6 \
                    -share_data_channels block3 block4 -data_and_control_channels 8
```

Case 3:

The design has one sharing group with 6 input channels shared between block1 and block2, but block3 does not share any inputs with other blocks.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 block2 -data_and_control_channels 6
```

Case 4:

In this example, a block is included in multiple channel sharing groups:

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 \
                    -channel 6 -share_data_channels block1 block2 \
                       -data_and_control_channels 6\
                    -share_data_channels block1 block3 -data_and_control_channels 8
```

The following error is generated:

```
Error: Block 'block1' is included in multiple channel sharing groups.
```

Case 5:

In this example, a block is added to a channel sharing group but the block does not share input channels with any other blocks in that group.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 -data_and_control_channels 6 \
                    -share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
Error: Block 'block1' is specified inside a channel sharing group, but
does not share input channels with any other block.
```

Case 6:

In this example, a block is specified inside more than one channel sharing group.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 6 \
                    -share_data_channels block1 block1 -data_and_control_channels 6 \
                    -share_data_channels block2 block3 -data_and_control_channels 8
```

The following error is generated:

```
Error: Block 'block1' is specified multiple times inside a channel sharing
group.
```

Case 7:

This example assumes that each block needs two control channels.

```
analyze compression -edt_block block1 -chain 100 -channel 3 \
                    -edt_block block2 -chain 200 -channel 4 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 6
```

The following error is generated:

```
Error: Channel sharing group with 'block1' (2 control channel), 'block2'
(2 control channel), and 'block3' (2 control channels) needs at least 7
input channels for channel sharing."
```

Case 8:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. Because the command specifies to share nine channels, no channels are actually shared.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 3 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 9
```

The following error is generated:

```
Error: Excessive input channels specified for channel sharing group which
includes 'block1'. This group requires between 6 and 8 input channels.
```

Case 9:

This example assumes that block1 has one control and one data channel; block2 has one control channel and two data channels; and block3 has two control channels and two data channels. The command specifies to share five channels which is not enough.

```
analyze compression -edt_block block1 -chain 100 -channel 2 \
                    -edt_block block2 -chain 200 -channel 3 \
                    -edt_block block3 -chain 300 -channel 4 \
                    -share_data_channels -data_and_control_channels 5
```

The following error is generated:

```
Error: The available input channels (5) specified for the channel sharing
group is not enough to drive the data input channels for block 'block2'
and 'block3'. This group requires between 6 and 8 input channels.
Alternatively, if you want to keep the shared channel count (5), you may
reduce the number of input channels for the following blocks:
'block1' may have up to 2 channels.
'block2' may have up to 2 channels.
'block3' may have up to 3 channels.
```

Case 10:

In this example, channel sharing is specified but neither the total number of input pins nor any channel sharing groups are defined. In this case, the tool defines the channel sharing group to be all EDT blocks and calculates the required number of input channels for those three blocks.

analyze compression -share_data_channels

The following message is generated:

```
// Sharing 8 input channels across block1 block2 block3.
```

Case 11:

In this example, channel sharing is specified and two channel sharing groups are defined. However, the total number of input pins is not specified. In this case, the tool calculates the required number of input channels for each of the two defined channel sharing groups.

```
analyze compression \
                    –edt_block block1 –chain 100 –channel 2 \
                    –edt_block block2 –chain 200 –channel 4 \
                    –edt_block block3 –chain 300 –channel 6 \
                    –edt_block block4 –chain 200 –channel 2 \
                    -share_data_channels block1 block2 \
                    -share_data_channels block3 block4
```

```
// Sharing 8 input channels across block1, block2.
// Sharing 10 input channels across block3, block4.
```

Related Commands

set_edt_instances
set_edt_options
set_fault_sampling
set_fault_type

analyze_control_signals

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

Mode: setup, analysis

Usage

analyze_control_signals [-Report_only | -Auto_fix] [-Verbose]

Description

Identifies and optionally defines the primary inputs of control signals.

The analyze_control_signals command identifies each control signal (clocks, set, reset, write-control, read-control, and so on) of every sequential element (DFF, latch, RAM, ROM, etc.) and optionally defines its primary input as a control signal. The purpose of analyzing the control signals is to identify the primary inputs that need to be defined as a clock, read-control, or write-control. This analysis also considers pin constraints, but only traces through simple combinational gates.

If the -Verbose option is specified, the tool issues messages that indicate why certain control signals are not reported. At the end of the analysis, the tool displays statistical information such as the number of primary inputs identified as control signals, their types, and additional information.

If the -Auto_fix option is specified, all identified primary inputs of control signals are automatically defined. For example, when a clock is identified, an implicit add_clocks command is performed to define the primary input. By default, all control signals are only reported.

This command performs the flattening process automatically, if executed prior to performing flattening.

Note

The tool defers to your expert knowledge when you use add_clocks to define a clock; the -Auto_fix option will not alter a clock definition specified by a preceding add_clocks command.

dft -scan Details

Because it is not possible to completely identify all control signals, you should only use the results of this command as a *starting point* in creating a dofile that defines the clock control signals, read and write controls in a design. You should use this dofile when the tool is being used for production scan and test logic insertion. Situations where the tool may not be able to identify particular control signals include the following:

- When the tool is unable to trace or simulate through complex logic, the tool cannot identify a top-level pin as a clock. These cases include clocks driven by PLLs or clock

gaters that require simulating a test setup procedure to obtain a sensitizable path to a top-level pin.

- The tool cannot accurately predict the off-state of a clock by observing the flip-flops it is driving and, therefore, requires the user to verify that the off-state of the clock/set/reset lines is correct.

Because of these types of issues, this command does not identify all clocks, their offstates, or controls signals. In this case, you should explicitly add the unidentified clocks using the add_clocks command.

Arguments

- -Report_only

  An optional literal that specifies to identify control signals only (does not define the primary inputs as control signals). This is the invocation default.

- -Auto_fix

  An optional literal that specifies to define the primary inputs of all identified control signals as control signals. For example, when a clock is identified, an implicit add_clocks command is performed to define the primary input.

- -Verbose

  An optional literal that specifies to display information on control signals (whether they are identified or not, and why) while the analysis is performed.

Example

The following example analyzes the control signals, then only provides a verbose report on the control signals in the design. After examining the transcript, you can then perform another analysis of the control signals to add them.

analyze_control_signals -verbose

```
//  command: analyze_control_signals -reports_only -verbose
-------------------------------------------------------------------------

//  Begin control signals identification analysis.
-------------------------------------------------------------------------
//  Warning: Clock line of '/cc01/tim_cc1/add1/post_latch_29/WRITEB_reg/r/
    (7352)' is uncontrolledat '/IT12 (4)'.
...
...
...
//  Identified 2 clock control primary inputs.
//     /IT23 (5) with off-state = 0.
//     /IT12 (4) with off-state = 0.
//  Identified 0 set control primary inputs.
//  Identified 1 reset control primary inputs.
//     /IRST (1) with off-state = 0.
//  Identified 0 read control primary inputs.
//  Identified 0 write control primary inputs.
-------------------------------------------------------------------------
```

```
//  Total number of internal lines is 105 (35 clocks, 35 sets , 35 resets,
      0 reads, 0 writes).
//  Total number of controlled internal lines is 25 (17 clocks, 0 sets ,
      8 resets, 0 reads, 0 writes).
//  Total number of uncontrolled internal lines is 80 (18 clocks, 35 sets,
      27 resets, 0 reads, 0 writes).
//  Total number of added primary input controls 0 (0 clocks, 0 sets ,
      0 resets, 0 reads, 0 writes).
-----------------------------------------------------------------------
```

analyze_control_signals -auto_fix

Related Commands

add_clocks
add_read_controls
add_write_controls
report_clocks
report_read_controls
report_write_controls

analyze_drc_violation

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

Mode: setup, analysis

Usage

analyze_drc_violation ***rule_id-occurrence#*** [-Display [APpend]]

Description

Generates a schematic in the Debug window of the portion of the design involved with the specified rule violation number.

When you issue the analyze_drc_violation command, the tool includes different simulation data in the netlist depending on the type of rule violation. The tool automatically opens the Debug window of DFTVisualizer if it is not already open, and sets the gate reporting to match the type of rule violation, just as if you had manually issued the set_gate_report command. The gate report setting is shown on the status bar. For some DRC violations (E5 for example), the gate level is automatically set to primitive.

Note that the following DRC errors do not have additional information that can be displayed with DFTVisualizer:

B1—B16
E1, E12, E13
G rules
K rules
P rules
T1, T8—T11, T13—T15, T18—T23
W1—W19, W32, W33

For more information about displaying DRC information, refer to the report_drc_rules command. For information about each DRC, refer to "Design Rule Checking" on page 2135.

Arguments

- ***rule_id-occurrence#***

  A literal and integer argument pair that specifies the exact design rule violation (including the occurrence) that you want to analyze. The tool traces the violation back to the probable cause and then the schematic viewer displays all the gates in that trace. The argument must include the design rules violation ID (*rule_id*), the specific occurrence number of that violation, and the hyphen between them. For example, you can analyze the second occurrence of the C3 rule by specifying C3-2. The tool assigns the occurrences of the rules violations as it encounters them, and you cannot change either the rule identification number or the ordering of the specific violations.

- -Display [APpend]

  A switch and literal pair that specifies to append the schematic to the existing contents of the Debug window instead of replacing the contents. If you specify the -Display switch without the Append switch, the existing schematic is deleted and new data is shown in the Debug window.

Examples

The following example defines the off-state of a clock incorrectly, causing a C2 rule violation. When a rule violation occurs, you can use the Debug window to analyze the probable cause of the error.

With this example, the Debug window displays the sequential element associated with the clk input, along with a backward trace through the gates and nets to the associated primary input.

add_clocks 0 clk
set_system_mode analysis

```
...
//  ---------------------------------------------------------
//  Begin scan clock rules checking.
//  ---------------------------------------------------------
//  1 scan clock/set/reset lines have been identified.
//  All scan clocks successfully passed off-state check.
//  Error: Clock /CLK cannot capture data with other clocks off. (C2-1)
```

analyze_drc_violation c2-1

Related Commands

open_visualizer
report_drc_rules
set_drc_handling
set_visualizer_preferences

analyze_fault

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Usage: analyze_fault path_name **{-Rise | -Fall}** [-Observe gate_ID#] [-Auto] [-Display [APpend]]

Stuck/Transition Usage: analyze_fault ***pin_pathname*** **{-Stuck_at {0 | 1} | -Rise | -Fall}** [-Observe *gate_id#*] [-Boundary] [-AUto] [-Continue] [-Display [APpend]]

Description

Performs an analysis to identify why a fault is not detected and optionally displays the relevant circuitry in DFTVisualizer.

The analyze_fault command performs an analysis to identify why the fault that you specify was not detected. You can use the -Observe switch to specify the observe point for the sensitization analysis.

If you are using DFTVisualizer, you can specify the -Display switch to graphically display the relevant circuitry for any fault detection. This may assist you in identifying either why a fault wasn't detected or how a fault was detected.

The fault analysis performed by the analyze_fault command consists of the following actions:

1. A message is given if the selected fault has been nofaulted.
2. A message is given that will identify if the fault is in the current fault list. If the fault is in the current fault list and the fault is the representative member, its fault classification is displayed.
3. If the fault was not identified as included in the active fault list, the basic fault analysis is performed that determines if the fault can be classified as unused, tied, blocked, or detected by implication.
4. If the fault category is detected by simulation, detected by implication, or unused, a message is given and the analysis is terminated. You can override the termination by using the -Continue switch.
5. If the fault category is tied, all sources of the tied condition are identified and the analysis is terminated.
6. If the fault category is blocked, all blockage points (100 maximum) are identified. For each blockage point, all sources of the tied conditions causing the blockage are identified. The analysis then terminates.
7. The states that result from all constrained pins and stable non-scan cells are calculated.

8. If the fault site is now prevented from attaining the necessary state, a message is given indicating the fault is tied by constrained logic, all sources of the tied condition are identified, and the analysis then terminates.
9. An analysis is made to identify all blockage points (100 maximum) and all potential detection points (25 maximum).
10. If there are no potential detection points, the blockage points are identified and for those points blocked by tied logic, all sources of the tied condition are identified. The analysis then terminates.
11. If there were potential detection points, the detection points are identified (25 maximum).
12. A controllability test generation is performed to determine if the fault site can be controlled. If successful, the test generation values are displayed using parallel_pattern 0. If unsuccessful, the analysis then terminates.
13. If an observe point is not selected, a complete test generation is attempted where the fault is sensitized from the fault site to any unblocked point. Potential problem points in any sensitization path are identified. The points will include tri-state driver enable lines, transparent latch data lines, clock_pos, wire gates, latch and flip-flop set/reset/clock lines, RAM write/read/address/data lines, and ROM read/address lines. If the test generation is successful, the test generation values are displayed using parallel_pattern 1.
14. If an observe point is selected, the fault is sensitized from the fault site to the observe point. Potential problem points in the sensitization path are identified. If the sensitization is successful, the test generation values are displayed using parallel_pattern 1.

Note

Only stuck-at, path delay and transition fault types can be analyzed using the analyze_fault command.

When the fault type is path delay, the analyze_fault command performs an unsensitizable path analysis when the ATPG run is unable to create a test pattern. If the analysis finds that some segment of the path is unsensitizable, it attempts to find a minimum number of gates in the path which are required to prove the path unsensitizable.

For example:

analyze_fault path37 -s 0

produces the following report:

```
//  ---------------------------------------------------------
//  Path delay fault analysis for path37 slow to rise on
// launch point = 22032, capture point = 167521
//  ----------------------------------------------------------
//  Path delay test generation not successful for capture point = 167521
//  due to atpg_untestable.
//  Begin unsensitizable path analysis for path=path37
//  Cycle=0 path sensitisation check failed status=redundant
//  Simultaneous sensitisation of 2 path segments cannot be achieved
//  (status=redundant).
//  These are:
//      OR   145279 (I1) + ;
//      ...
//      AND  150252 (I1) + ;
//  Path is a static unsensitizable path. Robust detection will be
//  impossible.
```

The analyze_fault report shows that it is impossible to use both input 1 (the second input, inputs are numbered starting at 0) of the OR gate 145279 while at the same time using input 1 of the AND gate 150252. The ellipsis (...) indicates that there are other gates in the path between the reported gates which are not relevant to the unsensitizable path problem.

The delete_fault_sites -Unsensitizable_paths command lets you delete proven unsensitizable paths.

When the fault type is transition, the analyze_fault command performs the following analysis steps:

1. The controllability of logic value 0 and 1 at the fault site is analyzed.

   If the analysis succeeds, the fault site is controllable. You can access the analysis results by setting the gate report type to be parallel pattern 0(2) to set logic value 1(0) at the fault site for a slow-to-rise fault, or logic value 0(1) at the fault site for slow-to-fall fault.

   If the tool aborts this step or it otherwise fails, the fault cannot be detected due to a lack of controllability at the fault site. The tool then terminates the analysis unless you force it to continue.

2. The ability to launch a slow-to-rise (0 -> 1) or slow-to-fall (1 -> 0) transition at the fault site is analyzed.

   If the analysis succeeds, the desired transition can be created at the fault site. You can access the analysis results by setting the gate report type to be parallel pattern 3.

   If the tool aborts this step or it otherwise fails, the fault cannot be detected because a transition cannot be launched at the fault site. The tool terminates the analysis unless you force it to continue.

3. The tool attempts to generate a test pattern for the transition fault being analyzed.

If successful, a test pattern exists to detect the fault. You can access the analysis results by setting the gate report type to be parallel pattern 1 and the analysis process terminates.

If the tool aborts this step or it otherwise fails, the tool will proceed to the next step.

4. The tool attempts to generate a test pattern for a stuck-at fault at the same fault site.

   If successful, there exists a sensitive path from the fault site to an observation point. The reason for aborting or failing to generate the test pattern for the original transition fault is that launching a transition at the fault site may block the fault propagation path. You can access the analysis results for the stuck-at fault by setting the gate report type to be parallel pattern 4.

Arguments

- ***pin_pathname***

  A required string that specifies the pin pathname of the fault where you want to perform the analysis.

- **-Stuck_at 0 | 1**

  A switch and literal pair that specifies the stuck-at fault value that you want to analyze. The stuck-at values are as follows:

  0 — A literal that specifies that the tool analyze the *pin_pathname* for a “stuck-at-0” fault.

  1 — A literal that specifies that the tool analyze the *pin_pathname* for a “stuck-at-1” fault.

- **-Rise**

  A switch that specifies a slow-to-rise transition fault.

- **-Fall**

  A switch that specifies a slow-to-fall transition fault.

- -Observe *gate_id#*

  An optional switch and integer pair that specifies the observe point for the sensitization analysis.

  gate_id# — An integer that specifies a gate identification number whose location you want to use as the observe point for the sensitization analysis.

- -Boundary

  An optional switch that specifies to display the boundary faults when analyzing an ATPG untestable, tied, blocked, or redundant fault.

- -AUto

  An optional switch, applicable only to path delay faults, that automatically determines how far along the path the tool can successfully propagate a path delay fault.

- -Continue

  An optional switch that forces the tool to complete the analysis of faults that have already been detected by the pattern set. This lets you inspect the generated pattern by using the report_faults command.

  If you do not specify this switch and the tool detects the fault category by simulation, by implication, or as unused, then the tool displays a message and terminates the analysis.

- -Display [APpend]

  An optional switch and literal pair that specifies for the tool to create a gate list relevant to the fault being analyzed and, if DFTVisualizer is open, to automatically update the schematic in the Debug window with the information; if the Append switch is specified the schematic is appended to the existing contents of the Debug window instead of replacing the contents.

  The schematic view contains annotations for the following cases as required:

  - Successful ATPG — All gates that sensitize the fault effects to an observable point are added to the gate list. The pins on these gates are annotated with the simulated value that results from the pattern that was created.
  - Uncontrollable Fault Site due to Forbidden Conditions — All gates that prevent the fault site from attaining the required state are added to the gate list. The pins on these gates are annotated with the constrain_value data.
  - Blocked Fault Site due to Forbidden Conditions — All gates that sensitize the fault effects to the blockage point and all gates that prevent the blockage points from attaining the necessary values are added to the gate list. The pins on these gates are annotated with the constrain_value data.

Example

The following example performs test pattern generation, then performs an analysis to determine why the tool did not detect a stuck-at-1 fault at pin /G:

set_system_mode analysis
create_patterns
report test data -class au

```
//  fault analysis summary of 3 faults
//  number faults blocked by constraints = 3
//  number faults connected to nonscan_latch = 3
//  number faults connected to clock/set/reset = 3
//  number faults unclassified = 0
```

report_faults -class au

```
type   code   pin_pathname
----   ----   ------------------------------------
  1     AU    /lat1/DL2/CLK1
  1     AU    /G
  1     AU    /lat2/DL2/CLK1
```

analyze_fault /G -stuck_at 1

```
// ---------------------------------------------------------
// Fault analysis for /G (4) output G stuck at 1
// ---------------------------------------------------------
// Current fault classification = AU (atpg_untestable)
// 1 potential observation points were identified:
//    Potential observe point: PO /V_VOUT (1968).
// Controllability justification was successful
      (data accessible using parallel_pattern 0).
// Pattern type: Basic_scan.
// The following potential detection path problems were id'd:
// Potential detection paths propagated through 1
//    transparent cell clock lines.
//  Test generation performed using all observable detection
//    points.
//  Fault was not successfully detected - fault status =
//    atpg_untestable.
```

Related Commands

delete_fault_sites
report_faults
report_testability_data

analyze_graybox

Context: patterns -scan

Mode: analysis

Usage

analyze_graybox [–no_reset] [–preserve_instances *instance_objects*]
[-max_sequential_levels {*number* | unlimited}] [-collect_reporting_data]

Description

Identifies the instances and nets to be included in the graybox netlist by automatically setting their in_graybox attributes to “true”. Subsequently, a “write_design -graybox” command writes out the graybox netlist for the marked instances and nets.

Before issuing the analyze_graybox command, you must place the design in external mode. That is, the output wrapper chains should be set up in non-capture mode, whereas the input wrapper chains are set up in both capture and shift (or hold or rotate) modes. At the end of the analysis, a summary of the identified instances displays in the session transcript.

The graybox analysis performs the identification by tracing backward from all primary output pins, wrapper chain sequential cell input pins and specified preserve instance input pins. In general, the backward tracing used by the graybox analysis considers a path sensitization that is based on the state-stability values obtained by simulating the pin constraints and test-setup procedures provided for the external test mode. However, a pure structural tracing method is used when tracing backward from preserve instance input pins.

You should exclude the scan-out pins of the core chains from the backward tracing to prevent marking internal (core) logic for graybox. You accomplish this by setting the ignore_for_graybox attribute of the scan-out pins to “true” using the set_attribute_value command.

The -preserve_instances option can be used to include arbitrary instances (the instantiations of modules or the instantiations of library models) in the graybox logic. The tool bypasses the graybox analysis within specified preserve module instances and automatically includes all of their sub-instances and nets in graybox. This allows you to overcome the limitations that the sensitized path based backward tracing imposes, such as not allowing complex test-setup procedures that initialize non-scan cells in clocks paths, limiting the sequential depth, etc.

One typical scenario to use the -preserve_instances option is when the design includes EDT logic driving the scan I/O pins of the wrapper chains. The entire EDT block can be specified as a preserve instance if graybox analysis cannot identify the EDT logic due to setup restrictions. To also include any depth of sequential pipeline stages on EDT channel pins, EDT Finder can be used along with graybox analysis. EDT Finder then identifies the EDT logic with the pipeline stages and automatically adds them as preserve instances for graybox analysis.

Arbitrary instances and arbitrary nets in the design can be marked for graybox netlist generation by manually setting their in_graybox attributes to “true” using the set_attribute_value

command. These attributes are reset to “false” by the analyze_graybox command unless you specify the –no_reset switch.

Note

Manually marking instances and nets for graybox netlist is not recommended because the tool does not automatically mark the nets and instances between such instances and the instances identified through the graybox analysis. Besides, the manual marking of instances and nets should be performed hierarchically to cover the parent instances and nets in order to properly synthesize them in the generated graybox netlist.

Currently, only Mux-DFF scan architecture is supported with the graybox functionality.

For more information about graybox concepts and procedures, refer to “Graybox Overview” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- –no_reset

  An optional switch that specifies not to reset the previously set in_graybox attributes of all design instances, and set additional ones identified by the graybox analysis. By default, the tool resets all in_graybox attributes when the command is issued without this switch.

- –preserve_instances *instance_objects*

  An optional switch with a Tcl object list that automatically includes instances in the graybox netlist without having to deploy the graybox analysis tracing routines on these instances and the sub-instances contained within. The connections to and from the preserved instances are still determined by the analysis to include in the graybox netlist. You can also declare the specified instance objects as blackbox module instances before the analysis.

- -max_sequential_levels {*number* | unlimited}

  An optional switch and integer or literal pair that limits the number of sequential levels (flip-flops) traced backward from a trace origin (a PO, a wrapper cell input pin, or a “preserve instance” input pin). When the sequential levels limit is reached, the tool stops collecting the graybox logic for the trace origin. Sequential cells that are part of wrapper chains or initialized to constant values (Tie-0/1) with an active set/reset pin are not counted as sequential levels when tracing backward. The “number” argument can accept a value of 1 or greater, where 2 is the default value when this switch is not specified with the command. The “unlimited” argument specifies to not limit the number of sequential levels during backtracing.

- -collect_reporting_data

  An optional switch that specifies to count the number of leaf instances (library model instances at design level) that are marked for each backward tracing origin, such as a PO, a wrapper cell input pin, or a “preserve instance” input pin (all input pins combined). This switch is required if you want to use the report_graybox_statistics command to report the

collected data. Note that collecting reporting data requires additional netlist tracing and could impact analysis execution performance.

Example

The following command performs graybox analysis on the current design. The module instance "tap_ctrl_ins" and all sub-instances within are marked for graybox logic without going through the graybox analysis.

```
analyze_graybox -preserve_instances tap_ctrl_ins -collect_reporting_data
report_graybox_statistics -top 10
```

Related Commands

report_graybox_statistics
set_attribute_value
write_design

analyze_restrictions

Context: dft -edt, patterns -scan

Mode: analysis

Usage

All Fault Types Except IDDQ Faults Usage:
analyze_restrictions [-Auto | -OLD] [-COnflict_source_analysis]

IDDQ Faults Usage:
analyze_restrictions [-INT_float_auto_except]

Description

Performs an analysis to automatically determine sources of the problem from a failed ATPG run.

The analyze_restrictions command reports the ATPG restrictions that caused a failed ATPG run. If you issue the analyze_restrictions command without an argument, the default is -Auto. The analysis may be lengthy depending on the number of restrictions. You can stop the analysis at any time by pressing Ctrl-C.

Arguments

- -Auto

  An optional switch that automatically performs the following step-by-step analysis:

  - **Step 1** — The tool identifies the type of constraints that failed the analysis.
  - **Step 2** — The tool analyzes the problem constraint types and finds the minimum constraints that prevent the tool from passing the analysis.
  - **Step 3** — Optional. The tool performs a detailed analysis of each constraint identified in Step 2. You perform this detailed analysis by using the -COnflict_source_analysis switch in conjunction with the -Auto switch.

  The set of constraints that conflict with the constraint are reported. The tool analyzes the constraints in the following order:

  1. dynamic ATPG constraints
  2. dynamic cell constraints
  3. static ATPG constraints
  4. static cell constraints
  5. named capture procedure
  6. bus floating checking
  7. bus contention (with the specified driver restriction)

8. port contention

- -OLD

  An optional switch that performs the analysis using functionality from releases prior to the 2007_2 release. It reports the ATPG restrictions that caused a failed ATPG run by category. These categories include ATPG constraints, IDDQ constraints, IDDQ checks, and contention checks. If the analysis determines bus contention is preventing success, a sequential analysis is performed. For more information on sequential analysis, see the -analyze_sequentially option of the analyze_bus command.

- -COnflict_source_analysis

  An optional switch that performs a detailed analysis (Step 3) of each constraint identified in Step 2 in the -Auto switch description. This option can increase the tool's runtime under certain circumstances.

- -INT_float_auto_except

  An optional switch that creates Iddq exceptions automatically for the "-int_float" Iddq check failure sites found during the analysis. Iddq exceptions are defined for failures that are the result of "-int_float" Iddq checks previously specified with the set_iddq_checks command. You can report the automatically added exceptions with the report_iddq_exceptions command or use the delete_iddq_exceptions command to remove unwanted exceptions.

Example 1

The following example analyzes a failed ATPG run in which a user-specified ATPG constraint is conflicting with a cell constraint and causes the ATPG to fail test generation.

create_patterns

```
//  Error: Current restrictions can NOT be simultaneously satisfied with
//  ANY abort limit. Status=atpg_untestable.
//  Halting run.  Suggest analyze_restrictions.
//  You can specify the -NOAnalyze option in command "create_patterns"
//  to force the run, but coverage will probably be low.
```

analyze_restrictions

```
//   -------------------------------------
//  | Step-1  Global constraint analysis |
//   -------------------------------------
//    Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
//   ----------------------------------------------------
//  | Step-2  Identifying minimum conflicting constraint |
//   ----------------------------------------------------
//  Following atpg constraints conflict with the rest of constriants:
//    atpg constraint: 1 dynamic  /FF3/Q (10)
//  You can use -conflict_source_analysis switch to analyze which
//  constraints conflict with the above constraints.
```

To perform detailed analysis of the conflict, then issue the analyze_restrictions command with the optional -conflict_source_analysis switch:

analyze_restrictions -conflict_source_analysis

```
//   -----------------------------------
//  | Step-1  Global constraint analysis |
//   -----------------------------------
//    Order-1: dynamic ATPG constraints, analysis=to_be_analyzed
//   ----------------------------------------------------
//  | Step-2  Identifying minimum conflicting constraint |
//   ----------------------------------------------------
//  Following atpg constraints conflict with the rest of constriants:
//    atpg constraint: 1 dynamic  /FF3/Q (10)
//   ------------------------------------------------------------------
//  | Step-3  Identifying conflict source of first problem constraint |
//   ------------------------------------------------------------------
//  Following constraint conflicts with problem constraint #1:
//    atpg constraint: 1 dynamic  /FF1/Q (8)
```

Example 2

The following IDDQ example analyzes a failed ATPG run, displays the results of the analysis, and automatically adds an Iddq exception for each ZVAL gate that failed in an “-int_float” Iddq check. Only part of the report, about Iddq failures, is shown.

analyze_restrictions -int_float_auto_except

```
...
//  With Iddq constraints and Iddq checks removed, PASSES ability to
satisfy restrictions.
//  Problem is with Iddq specific restrictions. Analyzing further to find
problems.
//  With Iddq constraints removed, and all Set Iddq Check options removed
except -intfloat, FAILS.
//  Must fix -intfloat option problems to generate ANY iddq test given
current restrictions & circuit.
//  Will attempt to locate and report specific -int_float violators.
//  No single bus is constrained to Z (or already Z).  Will analyze
further.
//  The following ZVAL gates cause -int_float to fail by themselves.
//    ZVAL gate /cpu/alu/U2392/ (54417).  ABORTed.
//    ZVAL gate /cpu/alu/U4177/ (63712).  ABORTed.
//  Will add_iddq_exceptions for -int_float failure sites.
...
```

Related Commands

add_iddq_exceptions
analyze_atpg_constraints
delete_iddq_exceptions
report_iddq_exceptions
set_contention_check
set_iddq_checks

analyze_simulation_mismatches

Context: patterns -scan

Mode: analysis

Usage

analyze_simulation_mismatches {**-AUto**
| {**-FAilure_file** ***failure_file*** [**-VCD_file** ***vcd_filename***]}
| {**-MISmatches** ***mismatch_bit***...} | ***Verilog_testbench_pathname*** }
[-Internal_patterns | -External_patterns]
{[-SImulation_script ***simulator_invoke_script***] | [-MAnual]}
[-SCan_test | -CHain_test | -ALl_test] [-HOST ***server_name***] [-Lib ***lib_name***]
[-TEstbench_for_vcd ***new_filename*** [-REplace]] [-SHift_check { ON | OFF }] [-VErbose]

Description

Analyzes simulation mismatches that occur between test patterns and a simulation performed by a specified external timing simulator.

By default, the simulation VCD file, test patterns, and design data are all analyzed and the mismatch sources are identified. Once the analysis is complete, you can use DFTVisualizer to graphically display the overlapping data and pinpoint the source of each mismatch.

Refer to "Analyzing Simulation Mismatches" in the *Tessent Scan and ATPG User's Manual* for complete information.

Note

All generated files are placed in the *work_dft_debug* directory inside your working directory. This directory is created if it does not already exist.

When using the analyze_simulation_mismatches command for a pattern with a P53 violation, the tool issues an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```

The problem of the P53 error is due to an undefined clock waveform in the used timeplate. Refer to the "Resolving P53 Errors for analyze_simulation_mismatches" for additional information on resolving this issue.

Arguments

- **-AUto**

  Required switch that automatically runs the entire simulation mismatch analysis on the currently loaded test patterns and design. This is the default behavior.

  The top-level design and Verilog library must be compiled in your work directory prior to starting the simulation mismatch analysis. The tool will search for a "work" directory within

the current working directory for these compiled files. Use the -Lib switch to specify a different directory.

The tool uses ModelSim as the Verilog simulator unless you specify a different simulator with the -Simulation_script switch.

- **-FAilure_file *failure_file***

  Required switch and string pair that specifies the pathname to an existing failure file to use for the simulation mismatch analysis. Using this switch bypasses Stages 1 and 2 of the analysis.

  -VCD_file ***vcd_filename***

  Optional switch and string that specifies the pathname of an existing simulation results VCD file for the mismatch analysis. This switch can only be used with the -FAilure_file switch. Using this switch bypasses the debug test bench simulation performed in Stage 4.

- **-MISmatches *mismatch_bit...***

  Required switch and repeatable string that specifies a list of mismatches to analyze instead of using a failure file.

  A *mismatch_bit* should be specified in one of the following formats:

  fail_pattern_index chain_name cell_index

  For example: "*1 chain5 20*" specifies the mismatch is on scan pattern 1, scan chain5, scan cell index 20.

  fail_pattern_index PO_name

  For example: "*2 xyz*" specifies the mismatch is on pattern 2 at PO signal *xyz*.

- ***Verilog_testbench_pathname***

  Optional string that specifies the pathname of an existing Verilog test bench for the invoking the old mismatch analysis. This option supports only ModelSim and performs an interactive communication to the Verilog simulator instead of dumping VCD data

- -Internal_patterns

  Optional switch that specifies the current internal pattern set for the mismatch analysis. This is the default.

- -External_patterns

  Optional switch that specifies the currently loaded external pattern set for the mismatch analysis.

Note

The serial test bench can take a long time to simulate for large designs. It is recommended that you analyze the parallel test bench first, then analyze the mismatches specific to the serial test bench.

- -SImulation_script ***simulator_invoke_script***

  Optional switch and string pair that specifies a simulation invocation script for the mismatch analysis. The script must invoke an external simulator and compile and simulate the test bench specified by the -PAtterns switch. By default, the analysis uses the simulator specified by the set_external_simulator command.

- -MAnual

  Optional switch to specify the "analyze simulation mismatch" functionality to stop at the step when the external Verilog simulator is needed. This option is useful when the Verilog simulation environment is complex and the simulation script can not be written directly. When specifying the step, the tool stops after saving Verilog test bench and gives an instruction for what to do next to complete the mismatch debugging flow. See Example 6.

Note

Note that the -MAnual and -SImulation_script switches are mutual exclusive.

- -SCan_test

  Optional switch that specifies that the tool only analyzes scan patterns. This is the default.

- -CHain_test

  Optional switch that specifies that the tool only analyzes chain patterns.

- -ALl_test

  Optional switch that specifies that the tool analyzes both scan and chain patterns.

- -Lib ***lib_name***

  Optional switch and string pair that specifies the pathname of the working simulation library. The working simulation library contains the compiled top-level design and any required parts libraries. By default, the tool uses the library named *work* in the same directory as the pattern test bench. For ModelSim applications, you must use this switch to specify a different library whether or not you enter the *vlib* command.

- -HOST ***server_name***

  Optional switch and string that specifies the name of a remote host on which to run the simulation portions of the mismatch analysis. The specified server must be accessible via rsh. For more information on rsh requirements, see "Multiprocessing Requirements" in the *Tessent Scan and ATPG User's Manual*.

- -TEstbench_for_vcd ***new_filename***

  Optional switch and string that specifies a file name for the debug test bench created in Stage 3 (default name *mentor_default.v_vcdtb.v*). The name of the simulation results VCD file is derived from the name specified by this switch. The default VCD dump file name created from the debug test bench is *mentor_default.v_debug.vcd*

- -REplace

  Optional switch that overwrites the *debug_testbench_filename* file if it already exists.

- -SHift_check { ON | OFF }

  Optional switch and literal that allows you to enable/disable checking for simulation mismatches that occur during the scan shift cycles. By default, the tool performs shift checking, but you might want to disable this feature for performance reasons after you are sure that shift operation is working properly.

  ON — Checks and reports if the scan shift cycle contains a mismatch between the Verilog simulation value and the ATPG expected value.

  OFF — Analyzes mismatches only on the capture cycles.

- -VErbose

  Optional switch that displays additional information about the simulation commands run in the background and the detailed mismatch data.

Example 1

The following example assumes the pathname of the ModelSim bin directory is defined by the PATH variable and that patterns are generated but not yet saved. This example demonstrates the automatic simulation mismatch analysis flow—see "Automatically Analyzing Simulation Mismatches" in the *Tessent Scan and ATPG User's Manual*.

The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates a Verilog library named *my_mismatch_work*, compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. DFTVisualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

Figure 7-8. analyze_simulation_mismatches Example

```
set_external_simulator vsim -c
write_patterns results/pat_p.v -verilog -parallel -replace
sys vlib my_mismatch_work
sys vlog -work my_mismatch_work my_design_source/my_design.v -v my_verilog_source/verilog_lib.v

Model Technology ModelSim SE vlog 5.5a Compiler 2001.04
-- Compiling module DFFS_scan
-- Compiling module my_top_level_module
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling module xor3
-- Compiling module mux21
...
-- Compiling module buff
-- Compiling module mgc_latch
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling UDP ud_dff
-- Compiling UDP ud_dlat

Top level modules:
        my_top_level_module

analyze_simulation_mismatch -auto -internal -lib my_mismatch_work
//New directory work_dft_debug has been created to store files for mismatch debug.
//   Previous simulation mismatch sources are deleted.
//-----------------------------------------------------------------------------------------
//Step-1  Creating Verilog testbench work_dft_debug/mentor_default.v with failure file
//dump.
//-----------------------------------------------------------------------------------------
//Step-2  Running Verilog testbench work_dft_debug/mentor_default.v to create failure file.
//-----------------------------------------------------------------------------------------
//Step-3  Loading failure file work_dft_debug/mentor_default.v.fail.
//-----------------------------------------------------------------------------------------
//Step-4  Creating Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v with VCD dump.
//-----------------------------------------------------------------------------------------
//Step-5  Running Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v to create VCD
//file.
//Running ModelSim to create VCD file work_dft_debug/mentor_default.v_debug.vcd
//-----------------------------------------------------------------------------------------
//Step-6  Loading VCD file work_dft_debug/mentor_default.v_debug.vcd.
//-----------------------------------------------------------------------------------------
//Step-7  Analyzing 25 internal scan patterns in parallel format.
//-----------------------------------------------------------------------------------------
//ID   instance (gate_id)  pattern-ix  time(ns)   seq_depth  ATPG_val  VCD_val mismatch_reason
//---- ------------------  ----------  ---------  ---------  --------  ------- ---------------------------
//   1 /ix1286/Y (364)      scan-1         1900           0         1        0 incorrect library cell "and
//   2  /ix1278/Y (265)     scan-15       17300           0         1        0 incorrect library cell "and


report_mismatch_sources -display debug data wave
```

Example 2

This example demonstrates using the -failure_file switch when you have previously done Verilog simulation. This example assumes you have run an initial simulation to verify test patterns, generated a failure file, and created a script to set up and run an external simulator on a separate server from the ATPG session. The following example runs a simulation mismatch analysis on the specified failure file using internal test patterns on a separate server.

```
analyze simulation mismatch -failure_file pat.fs.v.fail -simulation_script runsim
```

```
// Running script "runsim pat.fs.v.fail_vcdtb.v" from local host to create VCD file pat.fs.v.fail_debug.vcd.
// Debug 24 internal patterns, CPU time = 0.02 sec
...
//  Step-5  Analyzing 24 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//     1  /ix1286/Y (364)     scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  /ix1278/Y (265)     scan-15         17300      0          1        0     incorrect library cell "and4"
```

For information on the output of this command, see the report_mismatch_sources command.

Example 3

This example shows the results of using the -failure_file and -vcd_file switches and shows a simulation script “runsim” for ModelSim:

```
#! /bin/csh -f
#Expect netlist has been compiled to work directory, next line commented
#vlog ~design/netlist.v -v ~/library/lib.v
vlog ${1}
vsim ${2} -c -do "run -all" -sdfmax /${2}/design_inst=~/design/good.sdf
```

Note that in the previous example, ${1} is the Verilog testbench name. The simulation script is used twice, first for running the testbench with the initial pattern set to get the failing bits, and then a second run for the reduced testbench to get the VCD file. Therefore, the vlog script needs to take a parameter instead of fixed testbench name. The ${2} is the Verilog testbench top instance name, which may change when the testbench name changes, so the name needs to be a parameter rather than a fixed name.

The following example uses the “runsim” script for automatic simulation mismatch analysis. Assume that the design and Verilog library have been compiled in advance in the work directory. After creating patterns, execute the following command:

analyze_simulation_mismatches -simulation_script runsim

```
// Running script "runsim work_dft_debug/mentor_default.v_vcdtb.v" from local host to create VCD file
work_dft_debug/mentor_default.v_debug.vcd.
...
//  ------------------------------------------------------------------------------------------
//  Step-5  Analyzing 24 internal scan patterns in parallel format.
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//     1  /ix1286/Y (364)     scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  /ix1278/Y (265)     scan-15         17300      0          1        0     incorrect library cell "and4"
```

Example 4

The following example shows a simulation script for NCVerilog:

```
#! /bin/csh -f
ncsdfc ../data/good.sdf
#Expect netlist to be compiled, next line commented out
#ncvlog ~/design/netlist.v -v ~/library/lib.v
ncvlog ${1}

#The following two commands create sdfcmdfile for ncelab to load SDF file
echo 'COMPILED_SDF_FILE = "~/design/top.sdf.X",' >! sdfcmdfile
echo "SCOPE = ${2}.design_inst;" >> sdfcmdfile
ncelab worklib.${2}:module -sdf_cmd_file sdfcmdfile
ncsim worklib.${2}:module
```

Example 5

The following example shows how to run a simulation analysis on a remote host.

analyze_simulation_mismatches -auto -host flea01 -simulation_script runsim

```
//  Simulation mismatch source is deleted.
//  Running script "runsim mentor_default.v_vcdtb.v" from flea01 to create VCD file mentor_default.v_debug.vcd.
//  Debug 25 internal patterns, CPU time = 0.02 sec
...
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//    ID  instance (gate_id)  pattern-ix  time (ns)  seq_depth  ATPG_val  VCD_val  mismatch_reason
//  ----  ------------------  ----------  ---------  ---------  --------  -------  ----------------------------
//     1  /ix1286/Y (364)     scan-1           1900      0          1        0     incorrect library cell "and4"
//     2  /ix1278/Y (265)     scan-15         17300      0          1        0     incorrect library cell "and4"
```

Example 6

The following example shows using the -MAnual switch:

analyze_simulation_mismatches

```
// ----------------------------------------------------------------------
// Step-1  Creating Verilog testbench work_dft_debug/mentor_default.v
// with failure file dump.
// ----------------------------------------------------------------------
// Manual flow is specified by user. Please simulate Verilog testbench
// work_dft_debug/mentor_default.v to create failure file
// work_dft_debug/mentor_default.v.fail.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/mentor_default.v.fail" to generate reduced testbench
// with VCD dump.
```

Upon completion of this step, take the Verilog test bench *work_dft_debug/mentor_default.v* and perform the Verilog simulation to product the failure file *mentor_default.v.fail*.

Then, you issue the following command:

analyze_simulation_mismatches -fail work_dft_debug/mentor_default.v.fail -manual

```
// ----------------------------------------------------------------------
// Step-1  Loading failure file work_dft_debug/mentor_default.v.fail.
// ----------------------------------------------------------------------
// Step-2  Creating Verilog testbench
// work_dft_debug/mentor_default.v.fail_vcdtb.v with VCD dump.
```

```
// ----------------------------------------------------------------------
// Manual flow is specified by user. Please simulate Verilog testbench
// work_dft_debug/mentor_default.v.fail_vcdtb.v to create VCD file
// work_dft_debug/mentor_default.v.fail_debug.vcd.
// Then, "analyze simulation mismatch -fail
// work_dft_debug/mentor_default.v.fail -vcd
// work_dft_debug/mentor_default.v.fail_debug.vcd" to complete the
// analysis.
```

After this step, take the Verilog test bench *work_dft_debug/mentor_default.v.fail_vcdtb.v* and perform the Verilog simulation to product the VCD dump file *work_dft_debug/mentor_default.v.fail_debug.vcd*.

Finally, you use the following command to complete the task:

```
ATPG> analyze_simulation_mismatches -fail work_dft_debug/mentor_default.v.fail -vcd work_dft_debug/mentor_default.v.fail_debug.vcd
// ----------------------------------------------------------------------
// Step-1  Loading failure file work_dft_debug/mentor_default.v.fail.
// ----------------------------------------------------------------------
// Step-2  Loading VCD file
// work_dft_debug/mentor_default.v.fail_debug.vcd.
// ----------------------------------------------------------------------
// Step-3  Analyzing 2 internal scan patterns in parallel format.
// -- ------------------ ---------- --------- --------- -------- ------- ---------------------------------------
// ID instance (gate_id) pattern-ix time (ns) seq_depth ATPG_val VCD_val  mismatch_reason
// -- ------------------ ---------- --------- --------- -------- ------- ---------------------------------------
//  1 /u102/    (124)    scan-10    2100      0          0        X     incorrect loading during the main shifts
```

Related Commands

report_external_simulator
report_measure_cycles
report_mismatch_sources
write_patterns
set_external_simulator

analyze_test_points

Context: dft -test_points

Mode: analysis

Usage

analyze_test_points

Description

Specifies the test points analysis and generates a list of test points to be inserted by the tool.

Before performing the test point analysis, you can manually add test points using the add_control_points and the add_observe_points commands to add test points. You must do this prior to issuing the analyze_test_points command. Use the delete_test_points command to remove these manually added test points if you do not want the test points to be considered during analysis.

Note

When you have generated a list of test points with the “analyze_test_points” command, then these test points have to be deleted before you can modify the locations where test points are allowed. Any existing test points need to be deleted before issuing the command “add_notest_points” or “delete_notest_points” commands.

Note

Not all gates in the design will be considered by the tool for inserting test points. For various reasons test points will not be inserted on gates in the clock tree or in the scan path, gates in protected blocks, or gates where the user has explicitly requested that no test points should be inserted (via the “add_notest_points” command). If there are many locations where test points cannot be inserted the tool may not be able to find the best locations for test points. This may increase the number of test points that is needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20% of the gate-pins the tool will generate a warning. The warning specifies the number of these locations as well as the main reason(s) why test points cannot be inserted at these locations.

Arguments

None

Example 1

The following example specifies that a maximum number of 50 test points are to be generated for a design that will be tested with a set of 30000 random patterns. Then it performs the test points analysis based on these specifications. After the test points are reported to the screen they are inserted into the netlist:

```
set_system_mode analysis
```

set_test_point_analysis -total_number 50 -pattern_count_target 30000

analyze_test_points

report_test_points

insert_test_logic

Example 2

The following example generates 50 test points and performs the test point analysis. It then deletes all the existing test points, sets the attribute no_control_point to true and again performs test point analysis. After the test points are reported to the screen they are inserted into the netlist:

set_system_mode analysis

set_test_point_analysis -total_number 50 -pattern_count_target 30000

analyze_test_points

report_test_points

delete_test_points -all

set_attribute_value block1 -name no_control_point -value true

analyze_test_points

report_test_points

insert_test_logic

Related Commands

add_control_points

delete_test_points

analyze_wrapper_cells

Context: dft -scan

Mode: analysis

Usage

analyze_wrapper_cells

Description

Identifies shared and dedicated wrapper cells for the primary I/O ports.

This command launches the identification of the shared wrapper cells for the primary I/O ports that can be controlled with the set_wrapper_chains command. The user-specified clocks and scan-related I/O ports are automatically excluded from identification. The wrapper cells identified for the primary Inputs will be stitched into the separate Input wrapper chains, while the wrapper cells identified for the primary Outputs will be stitched into the separate Output wrapper chains. When the set_wrapper_chains command was not issued, the analyze_wrapper_cells command will perform the default wrapper cells identification similar to issuing the set_wrapper_chains command with no arguments.

When the analyze_wrapper_cells command is used in conjunction with the set_registered_io command or/and the “set_wrapper_chains –io_registration ON” command, the wrapper cells analysis will result in registration of the specified primary I/O ports or the primary I/O ports that have failed the shared wrapper cells identification with the dedicated wrapper cells to guarantee better test coverage of the circuit. This means that new scan cells will be added to all specified and failed primary I/O ports during scan insertion, subject to certain design considerations. The dedicated wrapper cells added to the primary Inputs will be stitched into the appropriate Input wrapper chains, while the dedicated wrapper cells added to the primary Outputs will be stitched into the appropriate Output wrapper chains.

When you use the set_wrapper_chains command, with or without the set_registered_io command, the identification includes the identification of existing sequential elements accessible via combinational logic from primary inputs or primary outputs as input/output registration cells (that is, shared wrapper cells). Thus, a shared wrapper cell serves a dual purpose of both functional mode, as well as the control and observe function during test. The advantage of shared wrapper cells is that less logic must be added and the external test mode (ExTest) can test the timing of the actual path.

The report_wrapper_cells command reports information about the identified wrapper cells for each primary I/O port subjected to wrapper cell identification.

Example

The following example illustrates the use of the analyze_wrapper_cells command:

```
set_wrapper_chains -input_sen se_in -output_sen se_out -input_gates_level 1 \
-output_gates_level 1 -io_registration on
set_system_mode dft

analyze_wrapper_cells
//  Number of targeted sequential instances = 6
//  Performing scan identification ...
//  Wrapper Cells Analysis ... (Inputs: 9, Outputs: 4)
//  Wrapper Cells Selection ...
//    5 cell(s) are selected for input wrapper chains.
//    0 cell(s) are selected for output wrapper chains.
report_wrapper_cells -verbose
-----------------------------------------------------------------------------------------------------------
Primary I/O  Max Logic  # Wrapper Cells  Wrapper     Wrapper  Clock     New           Reason For
Port         Level      Identified       Cells       Chain               Registration  Failed
             [1/1]      [256/256]        Identified  Type                Cell Added    Identification
-----------------------------------------------------------------------------------------------------------
in8 (I)      0          1                flop5       Input    clk       No            --
in2 (I)      1          2                flop3       Input    clk       No            --
                                         flop4       Input    clk
in1 (I)      0          2                flop1       Input    clk       No            --
                                         flop2       Input    clk
in3 (I)      0          0                new cell    Input    clk       Yes           Max Logic Level
in4 (I)      0          0                new cell    Input    clk       Yes           Combinational Logic Only
in5 (I)      0          0                new cell    Input    clk       Yes           Max Logic Level
in6 (I)      0          0                new cell    Input    clk       Yes           Combinational Logic Only
out1 (O)     0          0                new cell    Output   test_clk  Yes           Input Wrapper Cell
out2 (O)     0          0                new cell    Output   test_clk  Yes           Combinational Logic Only
-----------------------------------------------------------------------------------------------------------

The wrapper cells identification has failed for the following PIs and POs.
These PIs and POs will be I/O Registered unless excluded from the registration.
-------------------------------------------------------------------------------------------------------
in3                         Input      Exceeded the limit, 1, on the number of combinational logic levels
                                       between this PI and the first level of sequential cells
in4                         Input      Encountered combinational logic only before reaching PO: out2
in5                         Input      Exceeded the limit, 1, on the number of combinational logic levels
                                       between this PI and the first level of sequential cells
in6                         Input      Encountered combinational logic only before reaching PO: out2
out1                        Output     Encountered a sequential cell that was already identified as an
                                       input wrapper cell: flop4
out2                        Output     Encountered combinational logic only before reaching PI: in4
--------------------------------------------------------------------------------------------------------

These PIs and POs will be excluded from the I/O Registration unless they were explicitly included.
-----------------------------------------------------------------------------------------------
in7                                Input    Encountered no sequential cells and combinational logic
                                            other than buffers/inverters before reaching PO: out3
out3                               Output   Encountered no sequential cells and combinational logic
                                            other than buffers/inverters before reaching PI: in7
out4                               Output   Encountered no combinational logic and just 1 sequential
                                            cell before reaching PI: in8
-----------------------------------------------------------------------------------------------
insert_test_logic
```

Related Commands

report_wrapper_cells
set_registered_io
set_wrapper_chains

analyze_xbounding

Context: dft -scan

Mode: analysis

Usage

analyze_xbounding

Description

Performs X-bounding analysis.

This command is used to identify memory elements (current scan cells or cells that will be converted to scan-in a subsequent run) that might capture an unknown value during Logic BIST. In addition, the command identifies where to place bounding muxes that will prevent these X-sources from reaching the memory elements and corrupting the signature during Logic BIST.

The X-bounding analysis uses the E5 DRC to identify and bound all X sources from reaching scan cells and corrupting the MISR. All primary inputs and internal X source pins (non scan cells, blackbox outputs pins etc.) will be x-bounded.

The X-bounding process depends on pin constraints on the primary input pins to indicate them as X sources. These X-constraints are implicitly enforced by this command, but can also come from existing input constraints. When a pin is explicitly constrained to either 0 or 1, it will not be treated as an X source.

The mux and scan cell library models used for X bounding can be specified using the cell_type attribute in the ATPG library or by the add_cell_model command. If any of the required library cell models are not specified, an error message will be issued.

The analyze_xbounding command also marks as notest points the previously observable gate pins that are on the input side of the x-bounding muxes. In addition, an X-source that does not feed into any scannable flops (for example, it simply connects to a primary output pin) will not be bounded. Any previously observable gate-pins that are driven by that X-source will also be marked as notest points.

Note

You must issue this command to perform X-bounding analysis. The tool does not automatically perform the analysis when you issue the insert_test_logic command.

Arguments

None

Example

This example transitions from setup to analysis mode, triggers the X-bounding analysis, and then reports the locations of the X-bounding muxes:

```
set_system_mode analysis
```

analyze_xbounding

report_xbounding -verbose

Related Commands

insert_test_logic

read_sdc

report_notest_points

report_xbounding

set_xbounding_options

annotate_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Usage

annotate_diagnosis ***-directory_path*** [-force]

Description

Adds DFM and RCD data, if available, to existing diagnosis reports.

If there is already a DFM rule of the same name and type in the report, the tool replaces it. If there are already existing RCD feature statistics in the report, the tool replaces them.

Note

The set_diagnosis_options -include_dfm_rules and -include_rcd_constants switches do not affect the annotation process. Information is always preserved or added in the annotated diagnosis reports based on the content of the VDB or LDB that is open.

Arguments

- ***directory_path***

  A required string that specifies the directory path of a diagnosis report.

- -force

  An optional switch that specifies to bypass any issues with md5 compatibility that arise.

Example

The following example adds the DFM rules from the VDB to the flogs.ya diagnosis report.

```
open_layout design.ldb design.lddb
annotate_diagnosis flogs.ya
```

Related Commands

delete_dfm

import_dfm

append_to_collection

Context: unspecified, all contexts

Mode: all modes

Usage

append_to_collection ***var_name obj_spec*** [-unique]

Description

Adds one or more objects to a collection referenced by the specified var_name.

The append_to_collection command appends all elements in obj_spec to the collection referenced by the variable var_name. If the variable does not exist, the variable is created and made to point to a collection with elements from obj_spec as its value. An error is generated if the variable exists and does not reference a collection.

The result of the append_to_collection command is either a modification of the collection that was initially referenced by var_name or a new collection if the variable did not exist.

The append_to_collection command provides the same semantics as the add_to_collection command but with significant performance improvement. If you are building up a collection in a loop, using the append_to_collection command is more efficient as shown in Example 2.

Arguments

- ***var_name***

  A required value that specifies a Tcl variable name. The objects matching obj_spec are added to the collection referenced by this variable.

- ***obj_spec***

  A required value that specifies an object name, a Tcl list of object names, or a collection of objects to append to the collection referenced by var_name. The obj_spec can be an empty collection.

- -unique

  An optional switch that prevents duplicate objects from being appended to the collection. By default, duplicate objects are not removed.

Example 1

The following example appends a collection referenced by variable b to collection referenced by variable a.

```
set a [get_ports a*]
set b [get_ports b*]
append_to_collection a $b
```

Example 2

The following example creates a collection of all nets connected to pins of design instances matching u1/myPrefix*. Note that using the -unique option automatically removes duplicate

nets. This example is the same as the example shown for the add_to_collection command. Using the append_to_collection command for such an application is recommended and is much more efficient.

```
foreach_in_collection inst [get_instances u1/myPrefix* -of_type design] {
        set inputs [get_pins -of_instance $inst -direction input]
        set outputs [get_pins -of_instance $inst -direction output]
        append_to_collection sum [get_fanins $inputs -stop_on net] -unique
        append_to_collection sum [get_fanouts $outputs -stop_on net] -unique
}
```

Related Commands

add_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sizeof_collection

apply_specification_defaults

Context: dft, patterns

Mode: all modes

Usage

apply_specification_defaults [***id***] [-design_name *design_name*]

Description

Automatically applies all defaults specified in the DefaultSpecification/DftSpecification wrappers to the DftSpecification wrappers that were created manually.

You can also use this command when creating a DftSpecification wrapper in the Configuration Data window. You do not need this command when using the create_dft_specification command because the create_dft_specification command automatically applies the specified defaults while creating the DftSpecification wrapper.

Arguments

- ***id***

  An optional string that specifies the DftSpecification wrapper that is to be processed. As described in the DftSpecification wrapper syntax, the wrapper is identified by the two strings “design_name” and “id”. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(design_name,*), you can omit the id from the process_dft_specification command. However, if several DftSpecification wrappers matching DftSpecification(design_name,*) exist in memory, you must specify the appropriate id when executing the process_dft_specification command. Refer to the -design_name option to understand how the design_name value is obtained.

- -design_name *design_name*

  An optional switch and string pair that specifies the design_name value used to identify the DftSpecification wrapper. The -design_name value defaults to the current design if it has been set with the set_current_design command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

  When the id is specified, if exactly one wrapper matches “DftSpecification (*,<id>)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (*,*)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

Example

The following example calls the apply_specification_defaults command on a DftSpecification wrapper that was created manually to get the specified defaults found in the DefaultsSpecification/DftSpecification wrapper applied to the DftSpecification wrapper.

```
apply_specification_defaults
```

Related Commands

create_dft_specification

process_dft_specification

set_defaults_value

catch_output

Context: unspecified, all contexts

Mode: all modes

Usage

catch_output ***cmdline*** [-output *output_variable*] [-result *result_variable*] [-tee]

Description

Executes the specified tool command line, and may be used to prevent command errors from aborting the enclosing dofile or Tcl proc.

This command returns 0 if the embedded command succeeds and 1 if it fails.

You can use the -output switch to specify a variable to contain all of the output of the embedded command. You can use the -result switch to specify a variable to contain the Tcl result of the embedded command if the command returns one.

Arguments

- ***cmdline***

  A required string enclosed in braces "{ }" that specifies the command and associated arguments whose output you want to capture. The command is typically a report_* command. A return value of 1 indicates the command failed; a return value of 0 indicates the command succeeded.

- -output *output_variable*

  An optional switch and variable name that is set to the value of the entire command output.

- -result *result_variable*

  An optional switch and variable name that specifies to write the Tcl result returned by the embedded command to the result_variable variable.

- -tee

  An optional switch that specifies to write the command output to the transcript and to a logfile.

Example 1

The following example writes the result of the get_gate_pins command to the variable oVar; the result can be an error message if the command fails. If get_gate_pins fails, catch_output returns 1, and the "if" wrapper commands are executed.

The example also assigns the Tcl result returned by the get_gate_pins command to the rVar variable; the result is a collection if the command succeeds. If get_gate_pins succeeds, the catch_output command returns 0, and the "else" wrapper commands are executed.

The dofile continues even if the get_gate_pins command fails. Nothing is reported to the transcript.

```
if {[catch_output {get_gate_pins foo*} -output oVar -result rVar]} {
  # commands when get_gate_pins fails
} else {
  # commands when get_gate_pins succeeds
}
```

Example 2

The following example uses the catch_output command in a dofile to capture the output of a tool command in a variable.

```
set retVar [catch_output {report_gate /gate3} -output resultA -tee]
// /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//      Y         O  /ff5/D  ff4/D  ff3/D /gate6b/E

If { $retVar == 0 }  {
   puts $resultA
}
// /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//      Y         O  /ff5/D  ff4/D  ff3/D /gate6b/E
```

Related Commands

set_transcript_style

close_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

close_layout

Description

Layout-Aware Diagnosis Flow

Closes the Layout Database. Any subsequent diagnosis you perform is non-layout aware.

SVDB Layout Marker Flow

Releases the Calibre Query Server and closes the layout. If you issue this command, then the tool no longer generates layout markers.

Arguments

None

Related Commands

create_layout

open_layout

close_pattern_set

Context: patterns -ijtag

Mode: analysis

Usage

close_pattern_set [-network_end_state keep | reset | initial]

Description

Finalizes and closes the currently open pattern_set.

If any iRead or iWrite commands remain that have not been processed by an iApply statement, a warning is printed and iApply processes them automatically.

By default, the end state of the ICL network is the one achieved normally by the last iApply statement. When the -network_end_state option is specified with the reset value, the ICL network will be written to such that all network elements are back to their reset state at the end of the pattern_set. When the -network_end_state initial value is used, all network elements are restored to the state they had at the beginning of the pattern_set.

Arguments

- -network_end_state

  keep — No special actions are taken when closing the pattern_set and the state of the ICL network remains in the state reached at the end of the last iApply command. This is the default.

  reset — Value is used when it is necessary to restore the reset state of the ICL network at the end of the *pattern_set*. The restoring of the state only applies to registers controlling select lines of ScanMultiplexers; the other registers are not reset. The restoring of the state is performed iWrite commands to the state control registers. Depending on the ICL network configuration, the restoring of the reset state is often achieved without extra scan operations but some network topology may require that extra scan loads be used.

  You use this option when you need to run multiple patterns immediately after this one without doing an iReset between them. Because the reset state is restored at the end of the *pattern_sets*, you can run them in arbitrary orders.

  initial — Value used when it is necessary to restore the state of the ICL network at the end of the *pattern_set* to what it was at the beginning of the *pattern_set*. The restoring of the state only applies to registers controlling select lines of ScanMultiplexers. The restoring of the state is performed with iWrite commands to the state control registers. Depending on the ICL network configuration, the restoring of the initial state is often achieved without extra scan operations but some network topology may require that extra scan loads be used.

  You use this option when you need to run the same pattern_set multiple times in a loop and the pattern_set does not perform an iReset at the beginning. In this case, it is important that the ICL state be identical at the beginning of the first run as it is at the

beginning all subsequent runs. The pattern_sets that are typically run in a loop are patterns that perform some kind of polling and the polling routine.

Return Value

None

Example

```
open_pattern_set pat1
iCall inst1.mytest
close_pattern_set -network_end_state initial
```

Related Commands

get_open_pattern_set
get_pattern_set_list
open_pattern_set
report_pattern_sets
index_collection
reset_open_pattern_set

close_tsdb

Context: all contexts

Mode: all modes

Usage

close_tsdb ***directory_path_list*** | **-all** [-silent]

Description

Makes the content of previously opened TSDB directory invisible to the tool.

The specified TSDB output directory is automatically opened and cannot be closed with this command.

Arguments

- *directory_path_list*

  A value containing an absolute or relative path to a TSDB directory or a tcl list containing one or many absolute or relative paths to TSDB directories. You must specify either the directory_path_list or -all option, but you cannot specify both.

- -all

  An option to specify that all previously opened TSDB directory are to be closed. Note that the specified TSDB output directory is automatically opened and remains opened. You must specify either the directory_path_list or -all option, but you cannot specify both.

- -silent

  An optional switch that suppress the warning that is normally generated if the specified directory_path is not currently opened or if there are no opened TSDB directories when the -all option is used.

Examples

This example closes two TSDB directories with one command invocation.

```
> get_tsdb_list
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
> close_tsdb [list ${tsdb_home}/corea.tsdb \
                   ${tsdb_home}/coreb.tsdb ]
> get_tsdb_list
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

open_tsdb

get_tsdb_list

set_tsdb_output_directory

get_tsdb_output_directory

close_visualizer

Context: unspecified, all contexts

Mode: all modes

Usage

close_visualizer

DFTVisualizer Menu Path:
File > close_visualizer

Description

Saves display data and closes DFTVisualizer.

Display data is automatically saved and remains persistent for the current tool session.

Examples

The following example opens DFTVisualizer, displays the Debug window, and creates and displays a portion of the design, then closes the viewing session:

```
open_visualizer -display debug
add_display_instances i_16_7 -backward -end_point
close_visualizer
```

Related Commands

open_visualizer
write_window_contents
set_visualizer_preferences

compare_collections

Context: unspecified, all contexts

Mode: all modes

Usage

compare_collections ***collection1 collection2*** [-order_dependent]

Description

Compares the contents of two collections.

If the same objects are in both collections, the result is 0 (zero). If the objects in the collections are different, the result is a non-zero value. The order of the objects can optionally be considered by using the -order_dependent switch.

If two empty collections are compared, the compare_collections command considers them identical and returns a result of 0 (zero).

Arguments

- ***collection1***

  A required string that specifies the base collection for the comparison.

- ***collection2***

  A required string that specifies the collection to compare with collection1.

- -order_dependent

  An optional switch that causes the order of the objects to be considered when comparing the collections. The collections are considered to be different if the objects are ordered differently, and a non-zero value is returned.

Example 1

The following example shows a variety of comparisons.

```
set a [get_ports a*]
{a[3] a[2] a[1] a[0]}

set b [get_ports b*]
{b[3] b[2] b[1] b[0]}

set c [get_ports c*]
{}

#Same objects, same order
compare_collections $a $a
0

#Same objects, same order
compare_collections $a $a -order_dependent
0

#Different objects
compare_collections $a $b
1

#Different objects
compare_collections $a $b -order_dependent
1

#Two null collections
compare_collections $c $c
0

#Same objects, different order
compare_collections [add_to_collection $a $b] [add_to_collection $b $a]
0

#Same objects, different order
compare_collections [add_to_collection $a $b] [add_to_collection $b $a] -order_dependent
2
```

Related Commands

add_to_collection
append_to_collection
copy_collection
filter_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

compress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

LDBs created after v2013.3: compress_layout [-REplace]

LDBs created with v2013.3 and before: compress_layout ***layout_database_name*** [-REPlace]

Description

Compresses a Layout Database.

You must open the Layout Database using the open_layout command before performing the compression operation.

See "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -REPlace

  An optional switch for overwriting an existing file.

- ***layout_database_name***

  A required string that specifies the location to and the name of the Layout Database to be created during the compression operation.

Related Commands

create_layout
open_layout
uncompress_layout

compress_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Usage

compress_patterns [*passes_integer*] [-Reset_au] [-MAx_useless_passes *integer*]
[-MIn_elim_per_pass *number*] [-MUltiple_detection [-Order]]
[-MIXed Essential | Order [-Delta *percentage*] [-Limit *integer*]]

Description

Compresses patterns in the current test pattern set.

Note

Mentor Graphics recommends you use create_patterns for the smallest pattern set. The compress_patterns command is for specialized processes.

The compress_patterns command performs static pattern compression on the current test pattern set by repeating fault simulation for the patterns in either reverse or random order and selecting only those patterns required for detection. The *passes_integer* argument specifies the number of pattern compression passes. The first pattern compression pass runs in reverse order and then alternates between random and reverse for additional passes. If you do not specify a *passes_integer* argument, the tool performs only one compression pass.

This is a multiprocessing command for fault simulation. For more information about multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

If you specify the -Reset_au option, then when the tool performs pattern compression, it selects the AU faults for later fault simulation. If during pattern compression these AU faults simulate as possible-detected, the tool labels them as PU (possible-detected—ATPG_untestable), and they receive test coverage credit as possible-detected faults. If the number of pattern compression passes is greater than 1, the tool only performs the resetting of the AU faults for the first pass.

The compress_patterns command has a residual memory effect. The initial mode (reverse/random) is not fixed. Instead, it toggles back and forth, starting with the mode last used in the same run. The tool always starts with the reverse order at invocation.

If you include the -Multiple_detection switch, the static compression performed by this command will not simulate faults in the posdet (PD) fault class (PT and PU subclasses) and as a result, the tool may discard patterns that detect PD faults. Therefore, the test coverage reported after using this command switch may not be accurate if there are PD faults. To ensure you get the most accurate report of test coverage in this case, re-simulate the patterns using one of the following approaches:

- Enter the "set_system_mode analysis" command followed by the simulate_patterns command.
- Enter the order_patterns command with the -Simulate option.

Arguments

- *passes_integer*

  An optional integer that specifies the number of pattern compression passes. The default is 1 (performs only one compression pass).

- -Reset_au

  An optional switch that specifies fault simulation of AU faults.

- -MAx_useless_passes *integer*

  An optional switch and integer pair that specifies the maximum number of consecutive, useless (no eliminated patterns) passes the tool allows before terminating the pattern compression process. This command option has no effect on the pattern set if the number of passes (*passes_integer* argument) is smaller than the *integer* value of this switch. The default is the *passes_integer* value.

- -MIn_elim_per_pass *integer*

  An optional switch and integer pair that specifies the minimum number of eliminated patterns required in a single pass to continue the pattern compression process. If you specify this switch, you must enter a value greater than 0.

- -MUltiple_detection

  An optional switch that specifies for the tool to use a multiple detection based algorithm to carry out static pattern compaction. When you use this switch, the tool runs only one pass of compaction and discards patterns that detect only posdet (PD) faults.

- -Order

  An optional switch used with the -Multiple_detection switch to specify pattern ordering at the same time as static pattern compaction.

- -MIXed Essential | Order

  An optional switch and string pair that specifies to use a mixed compaction technique to compact the current internal pattern set. If you use the "Essential" option, the tool considers essential faults, those faults detected only once in the pattern set, as candidates for dynamic compaction. This option ensures that the patterns used to detect the essential faults are not dropped during pattern compression to keep the reached coverage.

  If you use the "Order" option, the tool selects fault candidates for dynamic compaction based on pattern set ordering.

Note

Using the "-Mixed Order" switch turns off pattern classification until you issue a "set_pattern_classification On" command.

Note

Mixed compaction can reduce the size of the current internal pattern set in most situations, but Mentor Graphics cannot guarantee that in every case the size of the pattern set after the tool performs mixed compaction will be less than the size of the pattern set before mixed compaction. Therefore, be sure to save the current internal pattern set before carrying out mixed compaction.

- -Limit *integer*

  An optional switch and integer pair that modifies the behavior of the -Mixed option. The -Limit switch specifies the maximum number of faults that the tool will attempt unsuccessfully to merge with the target fault pattern when dynamic compaction is involved. Without the -Limit switch, -Mixed causes the tool to use an adaptive algorithm to determine this number.

- -Delta *percentage*

  An optional switch and real number pair (between 0 and 100) that modifies the behavior of the -Mixed Order option. Normally, -Mixed Order selects fault candidates for dynamic compaction based on pattern set ordering. The -Delta switch causes the tool to consider as candidates for dynamic compaction, the specified percentage of the total number of faults instead of using pattern set ordering. The default percentage is 2.

Example

The following example compresses the generated test pattern set with two passes; the first pass is by reverse order and the second pass is by random order:

```
set_system_mode analysis
create_patterns
compress_patterns 2
```

Related Commands

add_processors
create_patterns
set_atpg_compression

copy_collection

Context: unspecified, all contexts

Mode: all modes

Usage

copy_collection ***collection***

Description

Duplicates the contents of an existing collection, resulting in a new collection. The base collection remains unchanged.

The only time you need to copy a collection is when you want to append to one copy while leaving the original copy unchanged. Copying a collection is equivalent to using add_to_collection with an empty collection as the second collection.

Arguments

- ***collection***

  A required string that specifies the collection to be copied.

Example

The following example shows the results of copying a collection and then appending to the copy. Notice how a2 is modified by append_to_collection, but a1 remains unaffected. The second reference to a2 called a3 sees the updated collection because it is simply a second reference to the same collection referenced by a2.

```
set a1 [get_ports a*]
{a[3] a[2] a[1] a[0]}

set a2 [copy_collection $a1]
{a[3] a[2] a[1] a[0]}

set a3 $a2

append_to_collection a2 [get_ports b[0]]
{a[3] a[2] a[1] a[0] b[0]}

puts [get_name_list $a1]
{a[3]} {a[2]} {a[1]} {a[0]}

puts [get_name_list $a2]
{a[3]} {a[2]} {a[1]} {a[0]} {b[0]}

puts [get_name_list $a3]
{a[3]} {a[2]} {a[1]} {a[0]}
```

Related Commands

add_to_collection
append_to_collection
compare_collections
filter_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

copy_module

Context: all contexts

Mode: insertion

Usage

copy_module ***obj_spec*** **-new_name** ***module_name*** [-silent]

Description

Creates an exact copy of a design module and assigns it a new name so that it can then be used as part of a create_instance and replace_instances.

The copy_module command returns a collection with one module object corresponding to the newly created module.

Arguments

- ***obj_spec***

  A required value that specifies a module name or a collection containing a single module object as returned by the get_modules command.

- **-new_name** ***module_name***

  A required switch and value pair that specify the name of the new module once it has been copied. If module_name already exists, the module is copied and "_integer " is appended to the end of the module_name where integer increments as each new module of the same name is added.

- -silent

  An optional switch that suppresses the error that is normally generated if obj_spec does not point to a single module that exists. A null collection is returned when an error was suppressed. This option also suppresses the error that is generated if the specified ***module_name*** already exists.

Example 1

The following example creates a copy of module ModA to ModB. ModB is identical to ModA and can be instantiated into the design.

copy_module ModA -new_name ModB

```
{ModB}
```

Example 2

The following example introspects the module name of an instance to create a copy.

copy_module [get_att_value u1/u2 -name module_name] -new_name ModB

```
{ModB}
```

Related Commands

get_nets
get_icl_ports
create_module
get_icl_pins
get_common_parent_instance

copy_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist

Usage

copy_simulation_context [-from *source_context_name*] [-to *target_context_name*]

Description

Copies the simulation values and forces from one simulation context to another.

The copy_simulation_context command copies the simulation values and forces from any predefined or user-defined simulation context to any user-defined simulation context.

Note that you must specify either the -to or the -from switch, or specify both switches.

Arguments

- -from *source_context_name*

  An optional switch and string pair that specifies the simulation context from which to copy the simulation values and forces. The source_context_name can be predefined (listed in Table 7-40 on page 1648) or user-defined. If you don't use this switch and you do supply the -to switch, the command uses the current simulation context as the source_context.

- -to *target_context_name*

  An optional switch and string pair that specifies the simulation context to which to copy the simulation values and forces. The target_context_name can be any user-defined context name that already exists and cannot be a predefined context name (listed in Table 7-40 on page 1648). If you don't use this switch and you do use the -from switch, the command uses the current simulation context as the target_context.

Example

The following examples show two ways to copy the simulation values of the predefined simulation context, stable_load_unload, to the user-defined simulation context, myContext:

```
SETUP> set_current_simulation_context myContext
SETUP> copy_simulation_context -from stable_load_unload
```

You can accomplish the same as the above with a single command:

```
SETUP> copy_simulation_context -from stable_load_unload -to myContext
```

Related Commands

add_simulation_context
delete_simulation_contexts
get_current_simulation_context
get_simulation_context_list
report_simulation_contexts
set_current_simulation_context

create_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Usage

create_capture_procedures [-Name *capture_procedure_name*]
{**-PAtterns** [Internal | External] [*pattern_index*...] |
{{**-Clock_sequence** ***clock_sequence*** | **-SAME_clocks_pulse**
{**SIngle_clock_per_procedure** | **GRoup_noninteracting_clocks_per_procedure**}
[*pulse_times*]}
[-FORce_pi [*cycle_number*...]] [-MAsk_po] [-SLow_cycles {***cycle_number***...}]
[-Load_cycles {***cycle_number***...}] [[-COndition *pinpathname value* [*cycle_number*...]]...]
[[-TImeplate [Fast | Slow] *timeplate_name*]...]
[{-EXT_Clock_sequence *clock_sequence* | -EXT_Same_clock_pulse *external_clock_name pulse_times*} [-EXT_Timeplate *timeplate_name*]]} [{>|>>} *file_pathname*]

Description

Creates named capture procedures based on clock sequences either derived from patterns or specified at the command line.

The create_capture_procedures command is used before or after pattern generation. This command creates the procedures in the internal memory only. You can save the procedure to a test procedure file using the write_procfile command.

For information on using named capture procedures, see "At-Speed Test Using Named Capture Procedures" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -Name *capture_procedure_name*

  An optional switch and string pair that specifies the name of the capture procedure to create. If there is already a procedure with this name, a suffix is added to the name specified. If multiple named capture procedures are created, the specified *capture_procedure_name* is used as the prefix of each name and a suffix is added. If -Name is not specified, the default name is *user_capture_proc1*, *user_capture_proc2*, and so on.

- **-PAtterns** [Internal | External] [*pattern_index*...]

  A switch that specifies which patterns to extract clock sequences from for the named capture procedures. The following options are available:

  [Internal | External] — An optional literal that extracts clock sequences from internal or external patterns. The default is internal if not specified.

  [*pattern_index*...] — An optional, repeatable string that extracts clock sequences from the patterns with the specified index. The pattern index starts at 1. By default, the tool extracts clock sequences from all the patterns of the internal or external pattern set if a pattern index is not specified.

- **-Clock_sequence *clock_sequence***

  A switch and string pair that specifies the clock sequence for the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or more clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The clock is designated by the gate identification number or pin pathname. For example, (c1,c2) (c1) (c2,c3) represents a clock sequence with three cycles. In the first cycle, clock c1 and c2 are pulsed; in the second cycle, clock c1 is pulsed; in the third cycle, clock c2 and c3 are pulsed.

- **-SAME_clocks_pulse** {**SIngle_clock_per_procedure** | **GRoup_noninteracting_clocks_per_procedure**} [*pulse_times*]

  A switch, literal, and optional value that automatically generate intra clock domain named capture procedures.

  SIngle_clock_per_procedure — A literal that pulses a single clock in every pulse cycle. A named capture procedure is automatically created for every clock in the design that is not constrained to its offstate value. Also, all clocks defined as its equivalence are pulsed when a single clock is pulsed in every cycle.

  GRoup_noninteracting_clocks_per_procedure — A literal pulses a clock group, which consists of a group of non-interacting clocks in every pulse cycle. Each clock group consists of a maximal set of non-interacting clocks. Once clocks are classified into groups, a named capture procedure is created for every clock group. (To understand the difference between non-interacting and interacting clocks, see the description of interacting clocks in the “domain_clock” and “-any_interaction” options in the set_clock_restriction command.)

  pulse_times — An optional value that specifies the number of cycles that pulse clock(s). By default, *pulse_times* is 2 if not specified.

- -FORce_pi [*cycle_number*...]

  An optional switch and repeatable integer that force PI values in the specified cycle(s). If *cycle_number* is not specified, all the cycles contain the force_pi statement. By default, only the first cycle contains the force_pi statement. The cycle number starts at 1.

- -MAsk_po

  An optional switch that specifies the last cycle contains the no measure_po statement. By default, the last cycle contains measure_po to indicate the time when the PO measurement happens.

- -Slow_cycles {*cycle_number*...}

  An optional switch and repeatable integer that specify the slow cycles of the procedure.

- -Load_cycles {*cycle_number*...}

  An optional switch and repeatable integer that specify where the scan load happens before the capture cycles of the procedure. By default, only the single scan load happens before the first cycle.

- -COndition *pinpathname value* [*cycle_number*]

  An optional switch, pin pathname, integer, and cycle integer that specify a condition of the scan cells in the named capture procedure. If *cycle_number* is not specified, the condition is a global condition; otherwise, the condition is a local condition associated with the specified cycle only.

- -TImeplate [Fast | Slow] *timeplate_name*

  An optional switch, literal, and string that specify the timeplate for the fast or slow cycles of the named capture procedure. By default, the specified timeplate applies to the fast cycles of the named capture procedure.

  If this optional switch is not specified, the default timeplate is used for the named capture procedures. You can define the default timeplate by specifying the Set Default_timeplate statement in the procedure file.

- -EXT_Clock_sequence *clock_sequence* | -EXT_Same_clock_pulse *external_clock pulse_times* [-EXT_Timeplate *timeplate_name*]

  An optional switch, string, and value that automatically create the external mode of the named capture procedures. This switch should not be used together with the **-Patterns** switch.

  -EXT_Clock_sequence *clock_sequence* — An optional switch and string pair that specifies the clock sequence for the external mode of the named capture procedure. *clock_sequence* is a sequence of cycles. A cycle is denoted by one or multiple clocks enclosed by a pair of parentheses. Adjacent clocks within a cycle must be separated by either white space or a comma. The designated clock(s) must be external clock(s).

  -EXT_Same_clock_pulse *external_clock pulse_times* — An optional switch, string, and value that specify to pulse an external clock in every cycle in the external mode. You can specify the external clock by its gate identification or pin pathname. *pulse_times* specifies the number of cycles that pulse the external clock.

  -EXT_Timeplate *timeplate_name* — An optional switch and string that specifies the timeplate to use for the external mode of the named capture procedure. The default timeplate is used if this optional switch is not specified.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file_pathname*.

Example 1

The following example creates named capture procedures based on the clock sequences extracted from internal patterns.

```
set_system_mode analysis
create_patterns
create_capture_procedures -name my -patterns internal
```

Example 2

The following example creates named capture procedures based on the clock sequences extracted from the 1st, 12th, and 20th patterns in the external pattern set.

```
read_patterns pattern_file
create_capture_procedures -name my -patterns external 1 12 20
```

Example 3

The following example creates a named capture procedure of two cycles for every non-constrained clock in the design.

```
create_capture_procedures -same_clocks single 2
```

Example 4

The following example creates a named capture procedure with a user-defined clock sequence. The defined clock sequence has four clock cycles. In cycle 1, clock1 is pulsed. In cycle 2, clock1 and clock2 are pulsed simultaneously. In cycle 3, clock1 and clock3 are pulsed simultaneously. In cycle 4, clock1 is pulsed and PO is measured. In this example, all cycles are at speed cycles.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1)
```

Example 5

The following example creates a named capture procedure using the same clock sequence defined in the previous example. In the first cycle, the scan cell needs to satisfy the condition of having the loading value of 1.

```
create_capture_procedures -clock_sequence (clock1) (clock1, clock2) (clock1, clock3) \
(clock1) -condition /inst_core/inst_iacc/inst_reg 1 1
```

Related Commands

delete_capture_procedures
report_capture_procedures
set_capture_procedures
write_procfile

create_connections

Context: all contexts

Mode: insertion

Usage

Usage 1: 1-to-1 pairing connections

create_connections ***object_spec1 object_spec2***
[-net_name *net_name*] [-silent]

Usage 2: 1-to-Many broadcast connections

create_connections ***object_spec1 object_spec2***
[-net_name *net_name*] [-silent]

Usage 3: Many-to-1 constant connections

create_connections ***object_spec1*** **-constant {1 | 0}**
[-net_name *net_name*] [-silent]

Description

Connects pin, net, or port objects.

You can connect any of the object types pins, nets, or ports as described here:

Usage 1: When ***object_spec1*** and ***object_spec2*** contain the same number of elements, the tool pairs the elements from the two lists in the order of the two lists and creates a connection between each pair. Only one driver element per pair is allowed.

Usage 2: When ***object_spec1*** has one element, and ***object_spec2*** has one or more elements, object_spec2 cannot include any driver elements, such as an input port or an output pin.

Usage 3: When you specify -constant, you cannot specify ***object_spec2*** and ***object_spec1*** cannot contain any driver element, such as an input port or an output pin.

When the -net_name option is not specified, the net and pin name used to create the connection is automatically inferred from the name of the elements within object_spec1 using the following algorithm:

- If the element in ***object_spec1*** is a pin, the tool uses the leaf_pin_name unless it has fewer than three characters; if it has fewer than three characters, the tool uses the leaf_instance_name of the pin concatenated with the leaf_name of the pin and separated by an underscore.
- If the element in ***object_spec1*** is a net or a port, the tool uses the leaf name irrespective of the number of character it has.

You can override the default net name by using the -net_name switch and specifying the net_name option to be used to create the intermediate nets and pins. Regardless of whether the default or a user-specified net name is specified, the tool checks for net conflicts and uniquifies

the newly created net to avoid accidental shorts. The uniquify operation is performed using the net_uniquification_string suffix specified with the set_insertion_options command.

The two endpoints of each connection may be in different parts of the design hierarchy. The tool creates all of the intermediate pins needed to create the connection through the hierarchy. The tool automatically reuses existing pins that already make up part of the requested connection.

If you connect more than one pin within a given instance to a common source outside the instance, the tool creates only a single pin on the instance interface and reuses that same pin for all of the desired connections specified within the instance.

Arguments

- ***object_spec1 object_spec2***

  A required value that specify a set of endpoints of the connections to be created. The value of the object_spec argument can be a single pin, port or net name; a Tcl list of pin, port, or net names; or a collection of pin, port, or net objects.

  The size of ***object_spec1*** must either be one or equal to the size of the ***object_spec2*** when ***object_spec2*** is specified. The ***object_spec1*** argument cannot include any driver elements when you specify the -constant option. At most, one driver can exist in each connection pair when you specify both ***object_spec1*** and ***object_spec2***.

  If an object does not exist within one of the **object_spec** and you specify the -silent option, the tool ignores the object and skips the connection request. If the object does not exist and you do *not* specify the -silent option, the tool generates an error.

- -constant 1 | 0

  A switch and literal pair that specifies a constant value to connect to all elements within ***object_spec1***. The -constant switch is mutually exclusive with the ***object_spec2*** argument.

  When this option is specified, the ***object_spec1*** argument cannot contain any driver elements such as input ports or output pins. In Verilog, a 1'b0 or 1'b1 connection is used as the connection to the pins, and a buf primitive or an assign statement with its input connected to 1'b0 or 1’b1 is used as the connection to output ports or to nets.

- -net_name *net_name*

  An optional argument that specifies the name of the pin created on ***object_spec1*** or ***object_spec2*** to route the connection. If –net_name is not specified, the tool uses the leaf name of object_spec1 to name the pin. If object_spec1 is a pin and its leaf name has fewer than three characters, the tool uses the leaf instance name of the base name of the pin concatenated with the leaf pin name and separated by an underscore.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages. Connections with at least one non-existent object are skipped.

Example 1

The following example illustrates Usage 1 and creates a connection between three scalar ports and a 3-bit bus pin. The pin name created on u2 to route the connection is u2_a[2:0]. This pin name is constructed by concatenating the leaf_instance_name of u1/u2/a with the leaf_name of pin a with an underscore, because it has fewer than three characters.

```
create_connections u1/u2/a[2:0] {a b c}
```

Example 2

The following example illustrates Usage 2 and creates three connections between port IN1 and pins blockA/modB/u1/a, blockA/modB/u2/a, and blockA/modC/u1/a. The command creates a single net called myname on u1, u1/u1 to make all three connections.

```
create_connections IN1 {blockA/modB/u1/a blockA/modB/u2/a blockA/modC/u1/a} \
–net_name myname
```

Example 3

The following example illustrates usage 3 and connects all pins matching the name ABC*/TE to a constant 0 value.

```
create_connections [get_pins ABC*/TE -hier] -constant 0
```

Related Commands

delete_connections
get_nets
get_icl_pins
get_icl_ports
move_connections

create_dft_specification

Context: dft

Mode: analysis

Usage

create_dft_specification [-existing_ijtag_host_scan_in *host_scan_in_design_pin_spec*] [-replace]

Description

Creates the "DftSpecification(design_name,id)" configuration wrapper and returns the newly created wrapper object so that it can be stored in a variable and used to customize the created specification.

This command finds Ijtag instances in the design that need to be connected to the IjtagNetwork and automatically creates the IjtagNetwork wrapper to connect them. The command supports multiple DFT insertion passes into any given module. The tool excludes elements seen to already be connected to the network from the current specification. An Ijtag instance is automatically seen by the command if its ICL file was loaded and the ICL module was matched to the design module during elaboration. See the "set_design_sources -format ICL" and the set_module_matching_options command descriptions for more details about this process. Use report_ijtag_instances or get_ijtag_instances to see which instances have a matched ICL module.

create_dft_specification.post_create callback procedure:

The DefaultsSpecification/DftSpecification wrapper has many options to help you customize how the created DftSpecification is created. You may however still have custom changes you want to implement on the created specification. You can easily automate these changes using the configuration introspection commands and setting commands to edit the "DftSpecification(design_name,id)" configuration wrapper as shown in Example 2 below.

The create_dft_specification.post_create procedure must have two arguments as shown here. The first argument holds the config wrapper name that is currently being created. The second argument holds a list of option value pairs corresponding to the options that were used when the create_dft_specification command was called.

```
proc create_dft_specification.post_create { root_wrapper args } {

  # "$root_wrapper" is the config_element object being created.
  # "get_config_value $root_wrapper -id 0" returns the design_name
  # "get_config_value $root_wrapper -id 1" returns the id

  # "$args" is a list of option value pairs that reflects the options used
  # when the process_dft_specification command was called.

  array set extra_args $args

  # $extra_args(-existing_ijtag_host_scan_in) returns the value of
  # that option
```

```
}
```

How the IjtagNetwork specification is created

The set_design_level command is used to define the target usage level of the current design. When the level is set to “chip”, the IJTAG interface created by the create_dft_specification command is a client Tap interface. When the level is set to “physical_block” or “sub_block”, it is a client IJTAG interface.

For the first dft insertion pass containing any IJTAG component, a root level IJTAG ring is created with one or two Sibs in series. The two sibs are called Sib(sti) and Sib(sri) and are inserted on demand when needed by the current insertion pass if it was not already inserted by a prior insertion pass. The Sib(sti) provides access to the IJTAG network for all instruments in the current design that are to be scan tested along with the rest of the functional logic by the logictest modes such as EDT or LogicBist. The acronym “sti” refers to “Scan Tested Instrument”. As shown in Figure 5-9 on page 145, that Sib provides isolation between its host port and its clients. The Sib(sri) is used to provide access to the IJTAG network for child physical blocks, and for logictest instruments and Tdrs used to configure the logictest modes. The acronym “sri” refers to “Scan Resource Instrument”. These IJTAG nodes must remain active and undisturbed during the logictest modes, and are excluded from the logictest scan chains. An ICL module is identified as a Scan Resource Instrument when it has the ICL module attribute called “keep_active_during_scan_test” set to “true”.

At the chip level, a TAP controller is inserted on the first insertion pass unless the -existing_ijtag_host_scan_in option of the create_dft_specification command has been used to reference a design pin. The design pin must be on an instance having an associated ICL module and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. Those different scenarios are illustrated in the Example 1 section below.

Any instance of a module having a corresponding ICL module will, by default, be inserted into the IjtagNetwork wrapper of the DftSpecification unless it is seen as being already connected to the network. The set_ijtag_instance_options command or the tessent_use_in_dft_ specification icl_module attribute can be used to change the default behavior.

Different ICL elements (ScanInterfaces, DataInPorts, and DataOutPorts) seen in the circuit are handled by this command as described here:

- Client ScanInterface — A Client ScanInterface is referenced by a DesignInstance wrapper when it is to be inserted into the network as part of the current DftSpecification. It is not referenced by the current IjtagNetwork when the ScanInPort associated to the Client ScanInterface has its attribute “has_functional_source” set to true. You can use the set_ijtag_instance_options command to force its inclusion or exclusion independent of the value of the has_functional_source attribute.

  If the ScanInterface is on an instance that is defined as a physical block (as defined by the tessent_is_physical_ module attribute of its ICL module), it will be inserted below the Sib(sri).

  If the ScanInterface is on an instance that is not defined as a physical block (as defined by the tessent_is_physical_ module attribute of its ICL module), it will be inserted

behind the Sib(sti) or Sib(sri) based on the value of the "keep_active_during_scan_test" attribute on its ICL module.

- DataInPorts — A DataInPort is referenced by a connection property inside the Tdr/DataOutPorts wrapper when it is to be controlled by the network. The DataInPort is not referenced by the current IjtagNetwork when there is already a DataOutPort in its fanin. If the DataInPort has a functional source, a multiplexer will be inserted to allow controlling it by the IjtagNetwork in test mode without disturbing its functional source its mission mode. You can use the set_ijtag_instance_options command to force its inclusion or exclusion independent of whether it has a DataOutPort in its fanin.

  If the instance on which the DataInPort exists has the keep_active_during_scan_test attribute defined and set to true, the Tdr sourcing the DataInPort will be inserted behind the Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the keep_active_during_scan_test attribute is off, the Tdr sourcing the DataInPort will be inserted behind the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

- DataOutPorts — A DataOutPort is referenced by a connection property inside the Tdr/DataInPorts wrapper when it is to be observed by the network. It is not referenced by the current IjtagNetwork when there is already a DataInPort in its fanout. You can use the set_ijtag_instance_options command to force its inclusion or exclusion independent of whether it has a DataInPort in its fanout.

  If the instance on which the DataOutPort exists has the keep_active_during_scan_test attribute defined and set to true, the Tdr observing the DataOutPort will be inserted behind the Sib(sri) so that it is kept active and not part of the scan chain during the scan test modes. If the keep_active_during_scan_test attribute is off, the Tdr observing the DataOutPort will be inserted below the Sib(sti) so that it can be scan tested along with the rest of the functional logic.

Licensing

This command normally requires an IJTAG product license to run. There is one exception to this requirement. If the design contains one or more instances of an EDT/LBIST controller, the TestKompress/LogicBist licenses are also allowed to run this command. In this case, the tool checks the availability of the license in the following order: IJTAG, TestKompress, LogicBist. If none are currently available, the tool queues up for the first license that you own.

You can refer to the set_license_queue_timeout and get_license_queue_timeout commands for more information about the license queuing feature.

Arguments

- -existing_ijtag_host_scan_in *host_scan_in_design_pin_spec*

  A optional option value pair that defines an existing host scan interface to which the entire IjtagNetwork should attach to inside the created DftSpecification. The design pin must be on an instance having an associated ICL module and the pin must be a ScanInPort referenced by a Host ScanInterface wrapper. You should only specify this option during the

first insertion pass. When unspecified, the interface to the ijtag network is determine using several criteria. When it is not the first insertion pass into the current design and either the Sib(sti) or the Sib(sri) is seen, it is automatically used as the connection point for the current insertion pass as illustrated in Step 2 of the example below. If it is the first insertion pass, a tap is inserted to host the root level Sib ring when design level is chip. The root level Sib ring is connected to ports when at the physical_block or sub_block level. The design level is specified as chip, physical_block or sub_block using the set_design_level command.

- -replace

  A optional switch that suppresses the error that normally occurs if the DftSpecification(design_name,id) configuration wrapper already exists in memory. Instead, the existing one is deleted and replaced by the newly created one.

Example 1

The following example illustrates the created DftSpecification that is generated by the create_dft_specification command when running on different modules of a chip. The examples below are useful to understand how instruments are connected to the IjtagNetwork in a multi-insertion pass scenario.

Step 1

The first run is performed on a core called CoreA to create and hook up instruments to the network. These instruments are scan testable (keep_active_during_scan_test is off). It is the first dft insertion pass involving the IjtagNetwork in that module so the tool creates the Sib(sti) in this run.

```
set_context dft -rtl
#commands to load design apprears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Sib(sti) is identified with the "tessent_dft_function" attribute set to "scan_tested_instrument_host". All local scan interfaces are daisy chained below the Sib(sti_local).

```
DftSpecification(CoreA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Sib(sti) {
        Attributes {
          tessent_dft_function : scan_tested_instrument_host;
        }
        Sib(sti_local) {
          DesignInstance(tdr_1) {
            scan_interface : client;
          }
          DesignInstance(tdr_1) {
            scan_interface : client;
```

```
            }
          }
        }
      }
    }
  }
```

Step 2

The second run is again performed on the core called CoreA at a later time to insert a Tdr used to control DataInBits on a scan control module that was user-inserted after the original insertion phase. The scan control module has an associated ICL file which documents its DataInPorts and with the ICL module attribute keep_active_during_scan_test set to on. Notice that because this is a second insertion pass into the same design module, the design_id is specified as rtl2 in the set_context command to differentiate the insertion pass from the previous one. An error would have been generated had it not been specified because a conflict would have been noticed from the existence of instances of module names whose names start with “CoreA_rtl_tessent_”.

```
set_context dft -rtl -design_id rtl2

#commands to load design apprears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. The Interface wrapper of the HostScanInterface wrapper is used to specify that the new Sib(sri) is to be inserted in series with the client interface of the Sib instance “CoreA_rtl_tessent_sib_sti_inst” which is the Sib(sti) created in the first insertion pass.

```
DftSpecification(CoreA,rtl2) {
  IjtagNetwork {
    HostScanInterface(sri) {
      Interface {
        design_instance : CoreA_rtl_tessent_sib_sti_inst;
        scan_interface : client;
      }
      Sib(sri) {
        Attributes {
          tessent_dft_function : scan_resource_instrument_host;
        }
        Sib(sri_local) {
          Tdr(sri_tdr1) {
            DataOutPorts {
              connection(0) : scan_control_inst/tm;
              connection(1) : scan_control_inst/ac_mode;
              connection(3:2) : scan_control_inst/config[1:0];
            }
          }
        }
      }
```

```
        }
      }
    }
```

Step 3

The third run is performed on the chip called ChipA. It is used to provide access to the IJTAG scan interface of the 2 instances of CoreA and to again insert a Tdr to control the data_in ports of the scan control module situated at the top level. A Tap wrapper is created as the -existing_ijtag_host_scan_in option was not specified and the Tap from a previous insertion pass was not seen as this is the first insertion pass at the chip level.

```
set_context dft -rtl

#commands to load design apprears here

set_current_design ChipA
set_design_level chip

set_system_mode analysis

set spec [create_dft_specification]
```

The generated DftSpecification is shown below. Notice how a Tap wrapper is being specified. It includes a HostBscan wrapper such that the TAP is ready for boundary scan-in a future insertion pass. One sib is inserted into the root level Sib ring to interface to the two instances of CoreA, and to provide access to the Tdr that is inserted to control the DataIn ports of the scan control module. Each child physical block is also wrapped by a local Sib to make sure other parts of the network are always accessible even if the physical block is defective, powered down, or in simulation, black-boxed.

```
DftSpecification(ChipA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Tap(main) {
        HostBscan {
        }
        HostIjtag(1) {
          Sib(sri) {
            Attributes {
              tessent_dft_function : scan_resource_instrument_host;
            }
            Sib(pb1) {
              DesignInstance(CoreA_I1) {
                scan_interface : ijtag;
              }
            }
            Sib(pb2) {
              DesignInstance(CoreA_I2) {
                scan_interface : ijtag;
              }
            }
            Sib(sri_local) {
              Tdr(sri_tdr1) {
                DataOutPorts {
                  connection(0) : scan_control_inst/tm;
```

```
                    connection(1) : scan_control_inst/ac_mode;
                    connection(2) : scan_control_inst/config2;
                  }
                }
              }
            }
          }
        }
      }
    }
  }
```

Step 3 could have been modified to connect to an existing TAP controller by simply using the -existing_ijtag_host_scan_in option as follows:

create_dft_specification -existing_ijtag_host_scan_in mytap/from_bist0

The created DftSpecification would then be as follows

```
DftSpecification(ChipA,rtl) {
  IjtagNetwork {
    HostScanInterface(ijtag) {
      Interface {
        DesignInstance (mytap) {
          scan_interface : BistPort0;
        }
        Sib(sri) {
          Attributes {
            tessent_dft_function : scan_resource_instrument_host;
          }
          Sib(pb1) {
            DesignInstance(CoreA_I1) {
              scan_interface : ijtag;
            }
          }
          Sib(pb2) {
            DesignInstance(CoreA_I2) {
              scan_interface : ijtag;
            }
          }
          Sib(sri_local) {
            Tdr(sri_tdr1) {
              DataOutPorts {
                connection(0) : scan_control_inst/tm;
                connection(1) : scan_control_inst/ac_mode;
                connection(2) : scan_control_inst/config2;
              }
            }
          }
        }
      }
    }
  }
}
```

Example 2

Below is an example of a Tcl proc that can be defined that will remove the HostBscan wrapper created inside the Tap wrapper. A wrapper called HostBscan is searched below the DftSpecification and deleted if it exists. The callback is automatically called inside the create_dft_specification command after the DftSpecification wrapper is created in memory.

```
proc create_dft_specification.post_create {config args} {
  set HostBscan_wrp [get_config_el HostBscan -hier -in $config -silent]
  if {[sizeof_collection $HostBscan_wrp] > 0} {
    delete_config_elements $HostBscan_wrp
    puts "post_create: Removing the HostBscan wrappers: [get_name_list \
          $HostBscan_wrp]"
  }
}
```

Related Commands

process_dft_specification

set_design_level

create_diagnosis_patterns

Context: patterns -scan_diagnosis

Mode: analysis

Usage

Logic Failure Diagnosis

create_diagnosis_patterns [-DIAGnosis_report {*diagnosis_report1 diagnosis_report2* ...}] [-MAX_pattern_count [*number*]] [-TRAnsition]

Scan Chain Failure Diagnosis

create_diagnosis_patterns [-DIAGnosis_report {*diagnosis_report*}] [-CHAIN *chain_name*] [-CELL_range *lowerbound upperbound*] [-FAULT_type {stuck_at_0 | stuck_at_1 | slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast}]

Description

Creates targeted patterns for logic failure or scan chain failure iterative diagnosis.

For usage cases using this command, refer to "Iterative Logic Diagnosis Examples" in the *Tessent Diagnosis User's Manual*. You can use iterative diagnosis with the layout-aware diagnosis flow as long as you use the layout in the same iterative diagnosis session—see "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual*.

Note

After issuing this command, you should write out the test patterns to an external file using the write_patterns command.

Logic Failure Diagnosis: The tool loads the diagnosis result from a prior run or a diagnosis report file, create a fault list, and generates the patterns targeting suspect signals in the diagnosis report.

Scan Chain Failure Diagnosis: The tool loads either of the following inputs you specify:

- A diagnosis report file containing the diagnosis results of scan chain failure.
- The chain names, scan cell range, and the fault type.

For scan chain failure diagnosis, you can also provide a chain name (using the –chain switch) and a diagnosis report containing multiple faulty scan chains. In this case, the diagnostic test generation only creates patterns for the scan chain specified by the –Chain switch.

Logic Failure Diagnosis Arguments

- -DIAGnosis_report {*diagnosis_report1 diagnosis_report2 ...*}

  An optional switch and one or more strings that specifies the diagnosis report file(s) to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. You can also supply more than one diagnosis report for suspects from multiple diagnosis report files are targeted in one diagnosis test generation run. The diagnosis report must be in the default ASCII format.

- -MAX_pattern_count [*number*]

  An optional switch and integer pair that specifies the maximum pattern count the tool generates. The default is no limit. Use when ATPG run time is a concern.

- -TRAnsition

  An optional switch that specifies creating transition test patterns. By default, the transition patterns generated are launch off capture (no_shift_launch).

Scan Chain Failure Diagnosis Arguments

- -DIAGnosis_report {*diagnosis_report*}

  An optional switch and string pair that specifies the diagnosis report file to be targeted by diagnostic ATPG. By default, the tool uses the diagnosis results from previous diagnosis run. Note that only a single diagnosis report is allowed as input for chain failures.

- -CHAIN *chain_name*

  An optional argument and string pair that specifies a single chain.

- -CELL_range *lowerbound upperbound*

  An optional argument and pair of integers that specify the beginning and ending of a range of cells.

- -FAULT_type {stuck_at_0 | stuck_at_1 | slow_to_rise | slow_to_fall | slow | fast_to_rise | fast_to_fall | fast}

  An optional argument and literal pair that specifies the fault type.

create_feature_statistics

Context: patterns -scan_diagnosis

Mode: analysis

Usage

create_feature_statistics

Description

Calculates the root cause deconvolution (RCD) constants from the LDB for RCD diagnosis and analysis.

The tool analyzes the design for all possible diagnosis suspects for certain features in the layout. It determines which bridge pairs, net segments, and cells are tested, and extracts the relevant physical information such as the critical area, via count, and cell instance count for each possible suspect.

You must create the RCD constants statistics for every flat model and pattern combination. To do this, you can process the combinations in parallel in different tool sessions.

RCD analysis in Tessent YieldInsight determines the probability sums for the features across the diagnoses and uses the RCD constants to normalize their weights to account for diagnosis noise.

To use this command, you must have previously loaded your patterns and opened a LDB. Otherwise, the tool issues an error message. The feature statistics are stored in the LDB for later use by Tessent YieldInsight RCD analysis. They are not used during the diagnosis process.

For more information, see “Diagnosis for Root Cause Deconvolution Analysis.”

Arguments

- None

Example 1

The following example loads patterns and opens the layout before calculating the feature statistics.

read_patterns external my_test_patterns.wgl

```
//  Reading WGL input file "my_test_patterns.wgl"
```

open_layout my_layout.db

```
//  DieArea: minx=-2107.000000, miny=-2002.000000, maxx=2107.000000,
maxy=2002.0
00000 in microns
//  Note: Checking layout consistency...
//  Note: Layout Database opened successfully.
```

create_feature_statistics

```
//  Determining feature detection information
```

```
//  10% Done.
//  22% Done.
……
//  100% Done.
//  Determining Open Feature Statistics
//  11% Done.
//  21% Done.
……
//  100% Done.
//  Determining Bridge Feature Statistics
//  13% Done.
……
//  Feature Statistics Calculation Complete. Information is written into
the layout database.
```

Example 2

The following example shows the message that the tool returns when you issue the create_feature_statistics when the feature statistics were previously calculated and stored in the LDB.

create_feature_statistics

```
//  Note: Feature statistics for this flat model and pattern set have
already been calculated and stored in the layout database. There is no
need to perform this operation again.
```

create_flat_model

Context: all contexts

Mode: setup

Usage

create_flat_model

Description

Creates a primitive gate simulation representation of the design. The command also flattens the hierarchy of the design so that the design can be viewed in the DFTVisualizer debug window.

The tool automatically flattens the design hierarchy down to the logically equivalent design when you exit Setup mode. However, there may be times that you would like to access the flattened model without having to exit Setup mode. For example, you may want to add atpg constraints and functions before you exit Setup mode.

If you exit Setup mode and then add atpg constraints and functions, the design rule checker does not have access to those ATPG constraints during the rule checking. If you issue the create_flat_model command in Setup mode and then add those ATPG constraints, the design rule checker has access to them during the rule checking.

Note

If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (issuing the create_patterns command) and saving final patterns. For more information, refer to "Preparing the Design Netlist" in the *Tessent Diagnosis User's Manual*.

Example

The following example shows flattening the design to the simulation primitives before adding constraints that the rule checker then uses when you run the design rule checker. The rule checker runs when you first attempt to exit Setup mode

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

The following example sets the system mode to setup, creates a flat model, and opens DFTVisualizer:

```
set_system_mode setup
create_flat_model
open_visualizer
```

Related Commands

set_system_mode
delete_atpg_functions
write_flat_model
add_atpg_constraints

create_initialization_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Usage

create_initialization_patterns ***RAM_instance_name*** | ***RAM_gate_id#***

Description

Creates RAM initialization patterns and places them in the internal pattern set.

The create_initialization_patterns command creates RAM initialization patterns that write values into the specified RAM. Tessent FastScan places these patterns into the internal pattern set from which you may save them into a file.

You can identify the RAM by its instance name or gate ID number. An error condition occurs if there is not a single RAM gate inside the instance or if the specified gate is not a RAM. An error condition also occurs if the RAM does not have an initialization file, or if the RAM did not successfully pass stability checking (Design Rules A1 and A6).

Tessent FastScan creates the initialization patterns by doing an independent test generation for each valid address which has a non-X state on at least one data line. If the test generation aborts, the command terminates with an error message. The patterns contain a measure PO statement, but all values are X. There may also be an unload statement, but all values are X. If the patterns are re-simulated (as in pattern compression), the tool deletes patterns that do not detect faults. For those the tool does not delete, simulation values replace the Xs in the measure PO statement.

Arguments

- ***RAM_instance_name***

  A string that specifies the instance name of the RAM for which you want to create_initialization_patterns.

- ***RAM_gate_id#***

  An integer that specifies the gate identification number of the RAM for which you want to create_initialization_patterns.

Example

The following example creates RAM initialization patterns for *p1.ram/u1*, places the patterns into the internal pattern set during the ATPG run, and saves the patterns to a pattern file with the name *patfile*:

```
add_write_controls 0 w1
set_system_mode analysis
add_faults -all
create_initialization_patterns p1.ram/u1
create_patterns
write_patterns patfile
```

Related Commands

read_modelfile

write_modelfile

create_instance

Context: all contexts

Mode: insertion

Usage

create_instance ***hierInstPath*** **-of_module** ***mod_spec*** [-allow_instance_uniquification] [-instance_uniquification_suffix *string*] [-parameter_list *parameter_value_list*] [-silent]

Description

Instantiates an instance of a module mod_spec inside a design module in the current design.

If the command is successful, this command returns a collection containing the newly created instance.

The pathname to the location must exist in the design up to the leaf level. If the leaf level already exists and the -allow_instance_uniquification switch is specified, Tessent Shell adds a uniquification suffix to the new instance to create a unique instance name. The uniquification suffix defaults to _# where # is replaced by the first available integer starting from 1. You can specify a different uniquification suffix using the -instance_uniquification_suffix option in the create_instance command or in the set_insertion_options command.

Arguments

- ***hierInstPath***

  A required string that specifies the hierarchical path in which the instance is to be created concatenated to the leaf instance name of the new instance. The specified hierarchical path in which the instance is to be created must exist in the design. If an instance of the module to be instantiated already exists in that module, by default the tool uniquifies the new name by adding a "_#" suffix to the instance name. You can use the instance_uniquification_suffix switch to specify a different suffix.

- **-of_module** ***mod_spec***

  A required switch and value pair that specifies the module to be instantiated. mod_spec can be the name of a module or a module object returned by the get_module command.

Note

Currently, any instantiation of parameterized modules can only be done with default values.

- -allow_instance_uniquification

  An optional switch that specifies to automatically uniquify the name of the new instance being instantiated if an instance of the same name already exists in the current design. This feature is very useful when inserting gates to intercept signals. You can always use the same leaf instance name and let the tool resolve conflicts automatically. This use model is illustrated in Examples 2, 3, 4 and 5 in the move_connections command description.

- -instance_uniquification_suffix *string*

  An optional switch that specifies a string to add to the new instance name when an instance of the same name already exists and the -allow_instance_uniquification switch is specified. If this switch is not specified, the tool automatically uniquifies the object using the default string "_#" where # begins at 1 and increments by one until a unique name has been found. The *string* must always end with a # to denote the uniquification integer.

- -parameter_list *parameter_value_list*

  An optional switch that specifies a Tcl list of parameter values when you use the command to create an instance of a parameterized module. The instance uses the supplied list of parameter values. For an example of how to use this switch, refer to "Example 4" on page 662.

  This switch is available only in -rtl mode ("set_context dft -rtl"). For more information, refer to the set_context command.

- -silent

  An optional argument that specifies to not generate an error message under the following condition. When the error is suppressed, no instance is created and the command returns a null collection. When no errors are suppressed, the command returns a collection containing the newly created instance.

  - The hierarchical path above the leaf instance name does not exist in the design.
  - The specified mod_spec is not the name of an existing module.
  - The collection supplied to the -of_module option does not contain a single module object.

Example 1

The following example creates an instance of module ModA called ModA_1 inside instance Block1_I1/Block2_I1. The return value is an instance object by the name of Block1_I1/Block2_I1/ModA_1.

create_instance Block1_I1/Block2_I1/ModA_1 -of_module ModA

```
{Block1_I1/Block2_I1/ModA_1}
```

Example 2

The following example creates two instances of module MUX called MUX inside instance Block1_I1. The tool uniquifies the leaf instance name when a name conflict occurs on the second invocation of the command.

create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification

```
{Block1_I1/MUX}
```

create_instance Block1_I1/MUX -of_module MUX -allow_instance_uniquification

```
{Block1_I1/MUX_1 }
```

Example 3

This example creates two instances of module MUX called MUX inside instance Block1_I1. The tool uniquifies the leaf instance name when a name conflict occurs. The get_attribute_value_list command returns the value of the leaf_name attribute on the new instance name.

**get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name**

```
{MUX}
```

**get_attribute_value [create_instance Block1_I1/MUX -of_module MUX \
-allow_instance_uniquification] -name leaf_name**

```
{MUX_1}
```

Example 4

This example creates an instance of a parameterized module using a list of parameter values. This example is based on the following Verilog module:

```
module ff(clk,d,q);
   parameter SIZE = 1;
   parameter OTHER = 0;
   input                clk;
   input   [(SIZE-1):0] d;
   output  [(SIZE-1):0] q;
   reg     [(SIZE-1):0] q;
….
}
```

Then the following command

create_instance ff_i3 -of_module ff -parameter_list { SIZE 10 }

creates the following Verilog code:

```
ff #(.SIZE(10), .OTHER(0)) ff_i3(.clk(), .d(), .q());
```

And this command

create_instance ff_i4 -of_module ff -parameter_list [list SIZE 10 OTHER 10]

creates the following Verilog code:

```
ff #(.SIZE(10), .OTHER(10)) ff_i4(.clk(), .d(), .q());
```

Related Commands

delete_instances
get_common_parent_instance
get_nets
get_icl_pins
get_icl_ports
rename_instance
replace_instances
uniquify_instances

create_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

create_layout ***layout_database_name*** {**-lef** ***lef_file_name*** … | **-leflist** ***leflist*** } {**-def** ***def_file_name*** ... | **-deflist** ***deflist*** } [-replace] {[-extra_layout_hierarchy *extra_levels_of_hierarchy_in_layout*] | [-extra_design_hierarchy *extra_levels_of_hierarchy_in_design*]} [-threshold *percentage_match*] [-temp_dir *directory_name*]
[-compression {on | off}] [-keep_uncompressed_database] [-verify_only]

Description

Generates a diagnosis tool-compatible Layout Database directory from LEF/DEF input files.

See "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

For LEF files, you must specify the file containing the layer definitions first followed by any of the MACRO definitions in any order.

The diagnosis tool performs a layout verification of the LEF/DEF files and reports any mismatches to the transcript—see "Layout Verification and Layout Database Creation Process in the *Tessent Diagnosis User's Manual*.

After the layout-verification stage, you can interrupt the Layout Database creation by issuing a **Ctrl-C** in the invocation shell. You can subsequently use the report_layout_rules command to obtain details on the layout rule violations and use this information to correct errors in LEF/DEF files, providing the correct LEF/DEF files and/or add any missing LEF/DEF files to the list provided to the create_layout command. Following this, you then use the create_layout command to determine if there are any additional issues. Use this method in iterative fashion to correct any problems with the LEF/DEF before creating the Layout Database.

Note

For maximum efficiency, you should create or use Layout Databases with the data on your machine's local disk.

Arguments

- ***layout_database_name***

  A required string that specifies the name of the Layout Database directory that diagnosis creates from the LEF/DEF input.

- **-lef** ***lef_file_name*** …

  Space-separated name(s) of the LEF file(s).

- **-leflist *leflist***

  A TCL list of the LEF file(s).

- **-def *def_file_name* …**

  Space-separated name(s) of the DEF file(s).

- **-deflist *deflist***

  A TCL list of the DEF file(s).

- -replace

  An optional switch for overwriting an existing file.

- -extra_layout_hierarchy *extra_levels_of_hierarchy_in_layout*

  An optional switch and string pair that removes extra levels of layout hierarchy such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if your design is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. Figure 7-9 illustrates this usage.

Figure 7-9. Extra Layout Hierarchy Example

In the case of Figure 7-9, the following command removes the hierarchy from the LEF/DEF net, pin names, and so forth before matching with the design name:

create_layout *layout_database_name* -lef *lef_file* -def *def_file* -extra_layout_hierarchy inst1

- -extra_design_hierarchy *extra_levels_of_hierarchy_in_design*

  An optional switch and string pair removes extra levels of design hierarchy such that the layout hierarchy and design hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if your layout is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces. Figure 7-10 illustrates this usage.

Figure 7-10. Extra Design Hierarchy Example

In the case of Figure 7-10, the following command removes hierarchy from a design pin, net name, and so forth before matching with layout:

create_layout *layout_database_name* -lef *lef_file* -def *def_file* -extra_design_hierarchy inst1

Note

The -EXTRA_DESign_hierarchy and -EXTRA_LAYout_hierarchy switches are mutually exclusive, and the tool issues an error if you specify both.

- -threshold *percentage_match*

  An optional switch and integer pair specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent creation of the layout database. The default threshold value is 85 percent. The following command sets the required layout match threshold to 92%:

  create_layout design.layout -lef lef_input.lef -def def_input.def -threshold 92

- -temp_dir *directory_name*

  In some cases, Tessent Diagnosis must update the Layout Database (for example, if you use the -verify switch). Due to the fact that the compressed Layout Database is a read-only database, Tessent Diagnosis automatically uncompresses, updates, and re-compresses the Layout Database. By default, the temporary uncompressed LADB is stored into the same location where the compressed Layout Database is located; if you have limited disk space in this location, then use the -TEmp_dir switch and string pair to specify an alternative location.

  See "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

- -compression {on | off}

  An optional switch and literal that controls whether the tool compresses the Layout Database during database creation. The default is OFF. Specify ON to create a compressed Layout Database.

  See "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

- -keep_uncompressed_database

  An optional switch that specifies that the tool retain a copy of the uncompressed Layout Database as well as a compressed version of the Layout Database.

- -verify_only

  An optional switch that enables LEF/DEF design verification.

Example 1

In the following example, the diagnosis tool creates a Layout Database (*design.layout*) from multiple LEF files and DEF files:

```
create_layout design.layout -lef lef_input01.lef lef_input02.lef -def def_01.def def_02.def
```

Example 2

In the following example, Tessent Diagnosis creates a Layout Database (*design.layout*) from a LEF file (*lef_input.lef*) and four DEF files (*a.def, b.def, c.def, d.def*).

```
create_layout design.layout -lef lef_input.lef -def a.def b.def c.def d.def
```

Related Commands

close_layout
compress_layout
open_layout
report_layout_rules
uncompress_layout

create_module

Context: all contexts

Mode: insertion

Usage

create_module ***module_name*** [-language {verilog | vhdl}] [-in_library *library_name*] [-vhdl_architecture *vhdl_architecture*] [-silent]

Description

Creates a new design module with no ports.

Ports can then be added to this new design module using the create_port command.

The create_module command returns a collection of one module object where the module is a newly created module.

Arguments

- ***module_name***

  A required string that specifies the name of the module to create.

- -language verilog | vhdl

  An optional switch and literal pair that specifies which language (Verilog or VHDL) to use when creating the module. The language defaults to Verilog when this option is not specified. The -language option is only allowed in the rtl context.

- -in_library *library_name*

  An optional switch and string pair that specifies in which library to create the module. The specified library_name must be a library name previously defined with the set_logical_design_libraries command. When the -in_library option is not specified, the module is created inside the default library which is also specified using the set_logical_design_libraries command. The -in_library option is only allowed in the rtl context.

- -vhdl_architecture *vhdl_architecture*

  An optional switch and string pair that specifies the name of the architecture to use for a created VHDL module. When -vhdl_architecture is not specified and -language is specified as vhdl, the created architecture is called struct. You cannot specify -vhdl_architecture unless -language is specified as vhdl. The -vhdl_architecture option is only allowed in the rtl context.

- -silent

  An optional switch that suppresses the error that is normally generated if module_name already exists. A null collection is returned when an error is suppressed.

Example 1

The following example creates a module called mytop and makes it the current module so that other modules can be instantiated in it.

set_ context dft -no_rtl
set_system_mode insertion
create_module mytop

```
{mytop}
```

set_current_design mytop

Example 2

The following example creates a module called myblock and appends the newly created module object to a collection referenced by the Tcl variable called created_modules.

append_to_collection created_modules [create_module myblock]

```
{myblock}
```

Example 3

The following example creates a VHDL module inside library lib1 with the default vhdl architecture struct. Notice how the get_modules command is used to report its attribute as report_attribute only looks for modules in the default library

create_module M1 -in_library lib1 -language vhdl

```
{M1}
```

report_attributes [get_modules M1 -in_library lib1]

```
Attribute Definition Report

Attributes defined for object 'M1':
Name                Value   Inheritance
-----------------   ------  -----------
file_path_name              -
is_created          true    -
is_modified         false   -
is_unsaved          true    -
is_valid            true    -
language            vhdl    -
library_name        lib1    -
line_number         0       -
master_name         M1      -
name                M1      -
object_type         module  -
type                design  -
vhdl_architecture   struc   -
```

Related Commands

copy_module
get_nets
get_icl_pins
get_icl_ports

create_net

Context: all contexts

Mode: insertion

Usage

create_net ***hier_net_name*** [-ascending] [-silent]

Description

Creates a net inside an instance of a design module in the current design.

This command returns a collection of created net objects. This command returns one object for each bit of a bus with the left bit first and the right bit last.

Arguments

- ***hier_net_name***

  A required string that specifies the path to the instance in which the net will be created, and the leaf name of the net to be created in the module. If the net name is a bus, this command creates one net for each bus bit, beginning with the left-most bit of the bus and ending with the right-most. The specified hierarchical path in which the net is to be created must exist in the design.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- -silent

  An optional argument that specifies the suppression of an error given one of the situations below. The command returns a null collection when an error is suppressed.

  - The hierarchical path of the net name specified does not exist in the design
  - The net already exist in the design

Example

The following example creates a two-bit bus net called A inside instance u1.

create_net u1/A[4:5]

```
{u1/A[4] u1/A[5]}
```

Related Commands

delete_pins
delete_nets
get_nets
get_icl_pins
get_icl_ports

create_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Usage

create_patterns [-OVerride_user_settings | -NO_AUto]
[-Compaction_effort {Low | High | Maximum | *integer*}] [-NO_Analyze]
[-COVerage_effort { Low | High }] [-NO_REDundancy_analysis]
[-NO_TERMINATE_ineffective_atpg] [-PATterns_per_pass *integer*]

Description

Performs automatic high-speed test generation and pattern compression.

This command is the recommended way to produce high quality, efficient, test pattern sets in the shortest possible time.

This is a multiprocessing command for ATPG. For more information about multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

The create_patterns command uses the built-in expert system, ATPG Expert, to create patterns. ATPG Expert analyzes the design and DRC and may modify tool settings to optimize test patterns for coverage, pattern count, and run time. By default, ATPG Expert only optimizes settings that have not been explicitly specified; it does not change user-specified settings. If you want ATPG Expert to also optimize user-specified settings, use the -override_user_settings switch. Additionally, a statistics report is automatically displayed. The create_patterns command displays the following statistics:

- Number of patterns simulated
- Test coverage
- Number of faults detected
- Number of faults remaining in the fault list (undetected)
- Number of effective patterns
- Number of test patterns
- Number of redundant, untestable, and aborted faults

  Occasionally, the display includes a line with three or four numbers separated by slashes (/). These numbers are, from left to right, the cumulative number of redundant (RE), ATPG_untestable (AU), and aborted faults. Aborted faults are separated into ATPG aborts (AAB) and EDT aborts (EAB) depending on whether compression is on. The sum of AAB and EAB aborts equal the total number of aborts.

  Example transcript without compression (these numbers are shown in bold font):

```
//#patterns test     #faults #faults   #eff.   #test    process    RE/AU/AAB
//simulated coverage in list detected patterns patterns CPU time
//   ---      ---     ---      ---      ---      ---    5.79 sec   0/40868/57

//    64     74.10%  135463  413980      63      63     14.73 sec
```

Example transcript with compression (these numbers are shown in bold font):

```
//#patterns   test   #faults #faults   #eff.   #test    process    RE/AU/AAB/EAB
//simulated coverage in list detected patterns patterns CPU time
//   ---     ------    ---     ---      ---      ---    5.79 sec   0/40868/27/30
//    64     74.10%  135463  413980      63      63     14.73 sec
```

At the end of the create_patterns run, the tool automatically performs redundant fault identification, which is equivalent to running the identify_redundant_faults command. If, however, you set any ATPG limit (for example, test coverage limit, run time limit, or pattern limit), then the tool does not perform redundant fault identification.

You should use the create_patterns command, review the transcript for any suggested command or setting changes, and implement the ones that help you achieve your test goals. Be aware that some of the suggestions have trade-offs (for example, greater run time in exchange for fewer test patterns); the merits of the trade-offs depend on your particular test requirements.

Note

When you perform compression analysis, the final order of scan cells in each scan chain in the design is not known. The order of the scan cells used as deterministic bits in the test cube impacts ATPG results. If many of the deterministic bits are clustered in consecutive cycles, more EDT_ABORT faults may appear because the clustered deterministic bits may not be compressed. When this occurs, ATPG test coverage may drop.

During compression analysis, the tool assumes the scan chains are organized in a balanced way based on an existing scan cell ordering. However, in the real design, scan cells may be in an arbitrary position depending on which scan stitching tool was used. If the clustering of deterministic bits occurs during compression analysis but not in the real design or if the clustering occurs in the real design but not in compression analysis, a discrepancy in the ATPG results may occur.

Note that you can use the add_faults command to define the scope of the create_patterns command. For more information, refer to “Example 3” on page 675.

The CPU time the tool reports is the time local to the command. If the command has multiple phases, then the reported CPU time is with respect to the beginning of that phase.

Arguments

- -OVerride_user_settings

  An optional switch that enables ATPG Expert to override user-specified settings if necessary to create an optimal test pattern set.

- -NO_AUto

  An optional switch that disables use of ATPG Expert to create patterns. During test generation, the default settings are used for those settings that have not been explicitly specified.

- -Compaction_effort {Low | High | Maximum | *integer*}

  An optional switch that specifies how much effort to spend trying to reduce the pattern count:

  Low

  Specifies a faster run time but a higher pattern count than the High option.

  High

  Specifies a longer run time but a smaller pattern count than the Low option. Use High when generating production patterns because this can significantly reduce test time and test data volume. This option usually gives the best results and is the default.

  Maximum

  Enables maximum tool effort to reduce pattern count. This option significantly increases run time and should be used only when absolutely necessary. Integer value 10 or 100 should be used before using this option due to the significantly long run time.

  integer

  A positive integer value that specifies compaction effort as follows:

  0 = Low
  1 = High (usually give the best results)
  n = The tool works *n* times harder than High

  As a general guide, when *n*=10, the run time increase is small, and the pattern count can be slightly better. When *n*=100, pattern reduction can be similar to Maximum, while the run time can be lower than Maximum. Differences in design complexity and size can affect which integer value provides the desired trade-off between pattern counts and run time.

  Note that you can significantly reduce the run time caused by higher integer values by using multiprocessing. For more information about multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual.*

- -NO_Analyze

  An optional switch that skips the analysis of test generation problems. Normally, before test generation, create_patterns checks for bus contention and whether ATPG restrictions and constraints can be satisfied. If these checks reveal potential test generation problems, an error message displays and the command terminates. If you want the analysis to be ignored and to proceed with test generation, reissue the command with this switch.

Note

Because this switch prevents normal analysis, all faults can be targeted, and ATPG effort expanded, yet without generating patterns. The extra effort may also give the appearance that the tool is stuck. Use this switch only in special cases when the risks are acceptable.

- -COVerage_effort { Low | High }

  An optional switch that improves ATPG test coverage. When set to High, the tool retargets aborted faults and uses dynamic learning during retargeting. Note that using this setting can result in significantly higher run times in some cases. By default, the tool does not retarget aborted faults.

  Using this switch may increase the ATPG run time.

- -NO_REDundancy_analysis

  An optional switch that, for ATPG using Named Capture Procedures, omits the redundancy identification analysis that follows pattern generation. Using this switch allows you to generate patterns faster. However, the test coverage calculations are less accurate without the redundancy identification analysis.

- -NO_TERMINATE_ineffective_atpg

  An optional switch that disables a feature that terminates the ATPG session after the patterns are no longer efficient at detecting faults. Note that using this switch results in a longer run time and may generate many additional patterns to detect very few faults.

- -PATterns_per_pass *integer*

  An optional switch and positive integer pair that specifies how many patterns to create and simulate in each pass. The integer value must be between 1 and 64. By default, the tool creates and simulates 64 patterns in each pass. When the value is 1, the tool generates and simulates every pattern, and so on. The switch affects only the current create_patterns session.

  The potential benefit of using a smaller integer value rather than the default 64 is better pattern count because ATPG is less likely to add non-essential faults into a pattern. However, when the value is small, the overall run time is likely to increase due to excessive fault simulation. Furthermore, a small value can reduce the run-time efficiency of multithreading and distributed ATPG.

  A smaller value is appropriate if the expected pattern count is small. A typical scenario for using a small integer value is IDDQ pattern generation, where you should use “create_patterns -patterns_per_pass 1” with “set_atpg_limits -pattern_count *n*” for creating *n* IDDQ patterns. You can use the -compaction_effort option with -patterns_per_pass to maximize the effectiveness of each pattern, but at the expense of additional run time.

Example 1

The following example creates, through ATPG Expert, an internally-stored set of optimally compact patterns for the entire netlist that can be saved using the write_patterns command.

ATPG Expert automatically decides the best setup (for example, sequential depth) for the design.

```
set_system_mode analysis
create_patterns
```

Example 2

The following example also creates an internally-stored set of compact patterns; however, ATPG Expert has been disabled. In this case, you need to specify by which sequential depth the pattern is created. The -sequential 2 switch means there are up to two clock pulses between scan loads. The patterns can be saved using the write_patterns command.

```
set_system_mode analysis
set_pattern_type -sequential 2
create_patterns -no_auto
```

Example 3

The following example creates patterns only for the specific block specified by the add_faults command:

```
set_system_mode analysis
add_faults /blockA/blockB
create_patterns
```

Related Commands

add_faults
identify_redundant_faults
reset_state
simulate_patterns
write_patterns
set_abort_limit
set_atpg_compression
set_atpg_limits
set_fault_type
set_pattern_type
set_random_atpg

create_pin

Context: all contexts

Mode: insertion

Usage

create_pin ***hierPinName*** [-direction input | output | inout] [-ascending] [-silent]

Description

Creates a pin on an instance of a design module.

If –auto_uniquify_edited_modules is On, the module is first uniquified before creating a pin on the specified instance. If –auto_uniquify_edited_modules is Off, the pin is created on the module and all instances of the module get the newly created pin.

This command returns a collection of the pin objects created; if hierPinName is a bus, this command returns a collection with one pin object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

Arguments

- ***hierPinName***

  A required string that specifies the hierarchical path of the pin to be created concatenated to the leaf pin name. The specified hierarchical path of the pin to be created must exist in the design.

- -direction input | output | inout

  An optional switch and literal pair that specifies the direction of the newly created pin. An input pin is created by default when this option is omitted.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- -silent

  An optional argument that specifies to not generate an error message if the specified hierPinName already exists or when the hierarchical part of the specified name is not an existing instance within the current design. When an error is suppressed, the command returns an empty collection.

Example

The following example creates a 5-bit bus input pin called A on the instance u1. The command returns a collection of newly created pin objects.

create_pin u1/A[5:1]

```
{u1/A[5] u1/A[4] u1/A[3] u1/A[2] u1/A[1]}
```

The following example introspects the direction attribute of the four newly created pin objects.

get_attribute_value_list [create_pin u1/C[3:0] -direction output] -name direction

```
{output output output output}
```

Related Commands

delete_pins
get_nets
get_icl_ports
get_icl_pins

create_port

Context: all contexts

Mode: insertion

Usage

create_port ***port_name*** [-on_module *obj_spec*] [-direction input | output | inout] [-ascending] [-silent]

Description

Creates a port on the specified design module.

By default, this command creates a port on the design module in the current design, If you specify the -on_module argument, this command creates a port on the module specified in *obj_spec*.

This command returns a collection of the port objects created; if port_name is a bus, this command returns a collection with one port object for each bus bit, beginning with the left-most bit of the bus and ending with the right-most.

If the specified port already exists the tool generates an error. You can specify the -silent switch to instruct the tool not to generate an error.

Arguments

- ***port_name***

  A required value that specifies the name of the port to be created. Bus ports must be created with a single create_port command where port_name includes the required bus range.

- -on_module *obj_spec*

  A required switch and value pair that specifies the module to which the port is to be added. The *obj_spec* can be the name of a module or a module object returned by the get_module command. If the -on_module switch is not specified, this command adds the port to the current design which is the module specified by the set_current_design command. You cannot create a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to create ports on modules created with the create_module command.

- -direction input | output | inout

  An optional switch and literal pair that specify the direction of the newly created port. An input port is created when this option is omitted.

- -ascending

  An optional switch that specifies the range direction type for a VHDL bus port. By default, the range direction type is inferred as “downto” when the left index is larger than the right index, and as “to” when the left index is smaller than the right index. You use this switch to specify that the VHDL port notation is “to” when the left index is equal to the right index; otherwise, the tool uses the “downto” range direction type. An error is generated if you

specify this option and the left index is larger than the right index, or if the object name is not a bus. The -ascending option is only allowed in the rtl context.

- -silent

  An optional argument that specifies to not generate an error message if the specified port_name already exists. When an error is suppressed, the command returns an empty collection.

Example 1

The following example creates a 5-bit bus input port called A on the module that is the current design. The command returns a collection of the newly created port objects.

create_port A[5:1]

```
{A[5] A[4] A[3] A[2] A[1]}
```

Example 2

The following example creates a scalar output port called B on module ModB.

create_port B -on_module ModB -direction output

```
{B}
```

Example 3

The following example creates a 4 -bit bus output port called C on the current design and uses the get_attribute_value_list command to query the direction attribute of the newly created port objects.

get_attribute_value_list [create_port C[3:0]] -name direction

```
{input input input input}
```

Related Commands

delete_ports

get_icl_ports

delete_atpg_constraints

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

Mode: setup, analysis

Usage

delete_atpg_constraints {***pin_pathname*** | ***gate_ID#*** | {***function_name*** | {**-Cell** ***cell_name*** {***pin_name***…}}}… [-Instance {*object_expression*…}] [-MODUle {*module_name* …}]}… | **-All**

Description

Removes state restrictions from the specified pins. You define pin restrictions by using the add_atpg_constraints command.

If you change ATPG constraints, create patterns with those changed constraints, and then compress patterns, all patterns not meeting the new constraints are rejected. This can cause the good patterns created with the old constraints to be rejected; therefore, use the delete_atpg_constraints command to remove all ATPG constraints before compressing the pattern set.

Arguments

- ***pin_pathname***

  A repeatable string specifying the pathname of the pin from which to remove any ATPG pin constraints.

- ***gate_ID#***

  A repeatable integer specifying the gate identification number of the gate from which to remove any ATPG pin constraints.

- ***function_name***

  A repeatable string specifying the name of a function created with the add_atpg_functions command and from which to remove any ATPG pin constraints.

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

  A repeatable switch and string pair that removes ATPG constraints from the specified pin on the DFT library cell. You can repeat the *pin_name* argument if there are multiple pins or nets on a cell that have ATPG constraints you need to remove.

- -Instance *object_expression*

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

- -MODUle *module_name*

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

- **-All**

  A switch that removes all current, user-defined ATPG constraints from all objects.

Example 1

The following example creates two user-defined ATPG pin constraints, runs the ATPG process, removes all ATPG constraints, and then compresses the pattern set:

```
set_system_mode analysis
add_atpg_functions and_b_in and /i$144/q /i$141/q /i$142/q
add_atpg_constraints 0 /i$135/q
add_atpg_constraints 1 and_b_in
create_patterns
delete_atpg_constraints -all
compress_patterns
```

Example 2

The following Tessent FastScan example shows how to use wildcards.

```
add_atpg_constraints 1 -cell mux21 Y -ins uALU/*/ix184 .
report_atpg_constraints .
```

```
1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
1 dynamic /uALU/u1/ix184/Y (4412) .
```

```
delete_atpg_constraints -cell mux21 Y -ins uALU/u1/* .
report_atpg_constraints .
```

```
1 dynamic /uALU/u3/ix184/Y (4400) .
1 dynamic /uALU/u2/ix184/Y (4406) .
```

Related Commands

add_atpg_constraints
add_atpg_functions
report_atpg_constraints

delete_atpg_functions

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

Mode: setup, analysis

Usage

delete_atpg_functions ***function_name***... | **-All**

Description

Removes the specified function definitions.

The delete_atpg_functions command lets you delete_atpg_functions defined with the add_atpg_functions command. You cannot remove an ATPG function if an ATPG constraint is currently using that function. If you attempt to remove an in-use function, the tool generates an error. Therefore, if you need to delete an in-use ATPG function, you must first remove all the associated ATPG constraints using the delete_atpg_constraints command; then you can remove the ATPG function.

You can display a list of the current ATPG functions that the tool is using as ATPG constraints by using the report_atpg_constraints command.

Arguments

- ***function_name***

  A repeatable string that specifies the names of the ATPG functions that you want to delete.

- **-All**

  A switch that removes all ATPG function definitions.

Example

The following example creates two user-defined ATPG functions, one user-defined ATPG constraint, displays the currently-in-use ATPG constraints, and then removes one of the inactive ATPG functions:

```
add_atpg_functions and_b_in And /i$144/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
add_atpg_constraints 0 /i$135/q
report_atpg_constraints

0 /$135/Q (23)

delete_atpg_functions and_b_in
```

Related Commands

add_atpg_functions
delete_atpg_constraints
report_atpg_constraints
report_atpg_functions

delete_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

delete_bist_capture_ranges ***capture_proc_name*** | **-All**

Description

Specifies that the tool no longer uses a particular named capture procedure when generating BIST patterns.

Arguments

- ***capture_proc_name***

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

- **-All**

  A switch that specifies that the tool no longer use any named capture procedures when generating BIST patterns.

Related Commands

add_bist_capture_range

report_bist_capture_ranges

delete_black_boxes

Context: all contexts

Mode: setup

Usage

delete_black_boxes **-Instances** [*ins_pathnames*] | **-Module**s[*module_names*] | **-All**

Description

Undoes the effect of the add_black_boxes command.

For a module that was originally modeled, removing the effect of the add_black_boxes command reinstates the original model. Specified tied values are no longer set on the output pins. For a module that was empty or undefined in the input netlist, the output pins revert to the default tied value of X.

Note

The tool releases the flattened model if one exists at the time you issue this command.

Arguments

- **-Instances** [*ins_pathnames*]

  A switch that specifies for the tool to undo the effect of the add_black_boxes command on all instances specified by the ins_pathnames argument. The ins_pathnames argument can be an instance, a Tcl list of instances, or a collection of instances. If you do not specify the ins_pathname, the tool undoes the effect of the add_black_boxes command on all instance-based blackboxes.

- **-Modules** [*module_names*]

  A required switch that specifies for the tool to undo the effect of the add_black_boxes command on all instances of modules specified by the module_names argument. The module_names argument can be an instance, a Tcl list of instances, or a collection of instances. If you do not specify the module_names argument, the tool undoes the effect of the add_black_boxes command on all module-based blackboxes.

- **-All**

  A required switch that specifies for the tool to undo the effect of the add_black_boxes command on all blackboxes.

Example 1

The following example adds the black box for module “core” then undoes all blackboxes that were defined.

```
add_black_boxes -modules core 1
delete_black_boxes -all
```

Example 2

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

```
delete_black_boxes -modules [get_modules -filter mark_black_box]
```

Related Commands

add_black_boxes
delete_tied_signals
report_black_boxes

delete_browser_data

Context: all contexts

Mode: setup, analysis

Usage

delete_browser_data {{**Gates** | **Primitives**} | {**Drc** [-All_fails | *rule_id*]} |
{**FAults** [-ALl | -Total | -Undet | *fault_class*]} |
F_coverage | **T_coverage** | **Relevant_Test_Coverage | Loss_TC |**
ATpg_effectiveness | **ALl}** [-Tab {Hierarchy | Library | Clocks}]

Description

Removes data from the active tab of the DFTVisualizer Browser window.

You use the -*Tab* switch to specify to remove the data column from a tab that is not the active tab.

Multiple arguments can be used in one delete_browser_data command.

Arguments

- **Gates**

  Required literal that deletes the design level gates listed for each instance.

- **Primitives**

  Required literal that deletes the primitive level gates listed for each instance.

- **Drc**

  Required literal that deletes DRC violation information from the Browser window. By default, all DRC violation information is deleted. Options include:

  rule_id
  String that specifies a design rule to delete violation information for. The *rule_id* must be the exact identification (ID) shown at the top of the column to delete. For a complete list of rule violation IDs, refer to the "Design Rule Checking" chapter.

- **FAults**

  Required repeatable literal that deletes fault information from the Browser window. By default all fault information is deleted. Options include:

  -Total
  Switch that deletes the display of total faults for all classes.

  -Undet
  Switch that deletes the display of numbers of undetected (UD) faults.

  fault_class
  String that specifies a fault_class to delete faults for. The *fault_class* must be the character string shown at the top of the column to delete. For a summary of available fault classes, refer to the report_faults command.

-ALl
Switch that specifies to delete all fault classes that are currently displayed. This switch is valid in the Hierarchy and Clocks tabs of the Browser window.

- **F_coverage**

  Required literal that deletes all the fault coverage data from the display.

- **T_coverage**

  Required literal that deletes all the test coverage data from the display.

- **Relevant_Test_Coverage**

  Required literal that deletes the test coverage for each submodule after untestable faults have been added/deleted from test coverage calculations.

- **Loss_TC**

  Required literal that deletes the undetected faults in an instance displayed as a percentage of the testable faults in the entire design.

- **ATpg_effectiveness**

  Required literal that deletes all the ATPG effectiveness data from the display.

- **ALl**

  Required switch that deletes all data from the display.

- -Tab {Hierarchy | Library | Clocks}

  Optional switch and literal that specifies the tab in the Browser window from which the column is to be deleted. By default, the delete_browser_data command removes data from the active window. You can use this switch to specify to remove the data column from a tab that is not the active tab.

Examples

The following example adds three columns to the active tab of the Browser window, and then removes the column that shows the number of C5 DRC violations:

```
add_browser_data drc c3 drc c5 drc c6
delete_browser_data drc c5
```

Related Commands

add_browser_data

delete_capture_handling

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

Mode: setup, analysis

Usage

delete_capture_handling {***object***... | **-All**} [-SInk | -SOurce]

Description

Removes the special data capture handling for the specified objects.

When you remove the special data capture handling, the default handling resumes. The default data capture handling specifies that the source elements pass on the values from the previous (not the current) clock cycle. When using the delete_capture_handling command, you must specify either an object name or all objects.

Arguments

- ***object***

  Specifies the object(s) for which you want the tool to remove any special data capture handling. The following lists the valid choices for the *object* argument:

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

  pin_pathname — A repeatable string that specifies the name of a pin within the design.

  instance_name — A repeatable string that specifies the name of an instance within the design.

  -Cell *cell_type* — A repeatable switch and string pair that specifies the name of a cell.

- **-All**

  A switch that removes all special data capture handling.

- -SInk

  An optional switch specifying that the *object* argument is a termination point. This is the default behavior for the command.

  If you use this switch in combination with the -All switch, the tool removes all special data capture handling on all the sink elements.

- -SOurce

  An optional switch specifying that the *object* argument is an origination point for data capture.

  If you use this switch in combination with the -All switch, the tool removes all special data capture handling on all the source elements.

Example

The following example changes the data capture handling of two specific gates and then removes one of those changes.

```
add_capture_handling new 1158 1485 -source
delete_capture_handling 1158
```

Related Commands

add_capture_handling

report_capture_handling

delete_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Usage

delete_capture_procedures {**-All** | **-Created_capture_procedures** | **-Loaded_capture_procedures** | **-Disabled** | ***capture_procedure_name***...}

Description

Removes the specified capture procedures from internal memory.

The delete_capture_procedures command deletes all or specific named capture procedures from the internal tool memory. A named capture procedure used by the current pattern set cannot be deleted.

Arguments

- **-All**

  A switch that deletes all the named capture procedures.

- **-CReated_capture_procedures**

  A switch that deletes all the named capture procedures automatically created by the tool.

- **-LOaded_capture_procedures**

  A switch that deletes all the named capture procedures loaded using the read_procfile command or from procedure files associated with the scan group definition.

- **-DIsabled**

  A switch that deletes all the disabled named capture procedures.

- ***capture_procedure_name***

  A repeatable string that specifies a named capture procedure to delete.

Example 1

The following example deletes all capture procedures.

```
delete cap proc -All
```

Example 2

The following example removes the capture procedure named *user_capture_proc1*.

```
delete cap proc user_capture_proc1
```

Related Commands

create_capture_procedures
report_capture_procedures
set_capture_procedures

delete_cdp_test

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

Mode: setup, analysis

Usage

delete_cdp_test ***test_name***

Description

Deletes a test and its associated test patterns from the CDP.

Arguments

- ***test_name***

  A required string that specifies the name of the test that you want to delete.

Example

The following example deletes test1 from the CDP.

```
delete_cdp_test test1
```

Related Commands

add_cdp_test

delete_cell_constraints

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

Mode: setup, analysis

Usage

delete_cell_constraints **-All** | ***pin_pathname*** | {***chain_name cell_position***} | {**-Drc C6**} | {**-CHain** ***chain_name***} | {**-CLock** ***clock_name***}

Description

Removes constraints placed on scan cells.

The delete_cell_constraints command deletes the constraints placed on scan cells using the add_cell_constraints command. You can specify a scan cell by using either a pin pathname or a position in a scan chain.

Arguments

- **-All**

  A switch that, in Setup mode, specifies to delete all constraints from all scan cells. In analysis mode (Tessent FastScan and Tessent TestKompress only), it specifies to delete all cell constraints that were added in non-Setup modes.

- ***pin_pathname***

  A string that specifies the name of an output pin which directly connects to a scan memory element (you can only specify output pins of buffers and inverters). Valid in Setup mode only.

- ***chain_name cell_position***

  A string and integer pair that specifies the name of a currently-defined scan chain and the position of the cell in the scan chain. The *cell_position* is an integer where 0 is the scan cell closest to the scan-out pin. Valid in Setup mode only.

- **-Drc C6**

  A switch and literal pair that specifies to delete all cell constraints added by “add_cell_constraints -drc c6”. Valid in analysis mode only.

- **-CHain** ***chain_name***

  A switch and string pair that specifies to delete all cell constraints added by the -Chain argument to the add_cell_constraints command. Valid in analysis mode only.

- **-CLock** ***clock_name***

  A switch and string pair that specifies to delete all cell constraints added by the -Clock argument to the add_cell_constraints command. Valid in analysis mode only.

Example

The following example deletes an incorrectly added cell constraint placed on a scan cell:

```
add_clocks 1 clock1
add_scan_groups group1 proc.g1
add_scan_chains chain1 group1 scanin1 scanout1
add_scan_chains chain2 group1 scanin2 scanout2
add_cell_constraints chain1 5 c0
add_cell_constraints chain2 3 c1
delete_cell_constraints chain2 3
add_cell_constraints chain2 4 c1
report_cell_constraints
```

Related Commands

add_cell_constraints

report_cell_constraints

delete_cell_library

Context: unspecified, all contexts

Mode: setup

Usage

delete_cell_library

Description

Removes all current cell libraries, the current design including the flattened model, and all tool data that is design-specific.

This command also closes any open DFTVisualizer windows.

When this command is deleting a flat model, and that flat model was saved with a list of enabled/disabled commands, then the restriction on the disabled commands is removed. In other words, all commands allowed in the tool will be enabled for the next design.

Arguments

None

Example

The following example removes all current cell libraries:

```
delete_cell_library
```

Related Commands

delete_design
read_cell_library
read_flat_model
read_verilog
set_current_design
set_design_macros

delete_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

delete_cell_models ***dftlib_model***… | {**-Type** {**INV** | **And** | **Buf** | **OR** | **NAnd** | **NOr** | **Xor** | **Mux** | **Scancell** | **DFf** | **LATCH**}} | **-All**

Description

Specifies the name of the DFT library cell that the tool is to remove from the active list of cells that the user can access when adding test points or that the tool can access when inserting test logic.

You originally added the cells to the active list with either the add_cell_models command or with the *cell_type* library attribute. If you remove a cell model from the active list, you only remove the cell from that list and do not change the DFT library.

If you accidently delete a DFT library cell from the active list with the delete_cell_models command, you can add the specified cell back into the active list with the add_cell_models command.

Arguments

- ***dftlib_model***

  A repeatable string that specifies the names of the particular DFT library models that you want the tool to remove from the active list.

- **-Type INV** | **And** | **Buf** | **OR** | **NAnd** | **NOr** | **Xor** | **Mux** | **Scancell** | **DFf** | **LATCH**

  A switch and argument pair that specifies the cell model type of all DFT library models that you want the tool to remove from the active list. The valid cell model types are as follows:

  INV — A literal that specifies a one-input inverter gate.

  And — A literal that specifies a two-input AND gate.

  Buf — A literal that specifies a one-input buffer gate.

  OR — A literal that specifies a two-input OR gate.

  NAnd — A literal that specifies a two-input NAND gate.

  NOr — A literal that specifies a two-input NOR gate.

  Xor — A literal that specifies a two-input exclusive OR gate.

  Mux — A literal that specifies a 2-1 multiplexer.

  Scancell — A literal that specifies a mux-scan D flip-flop.

  DFf — A literal that specifies a D flip-flop.

  LATCH — A literal that specifies a D latch.

- **-All**

  A switch that removes all cell models from the active list, including those tagged in the DFT library with the *cell_type* attribute. This switch does not change the contents of the DFT library, only the active list within the tool.

Examples

The following example removes a DFT library model from the active list:

```
add_clocks 0 clk
set_test_logic -set on -reset on
set_system_mode analysis
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
delete_cell_models or2
insert_test_logic
```

Related Commands

add_cell_models
report_cell_models
set_test_logic

delete_chain_masks

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

Mode: setup, analysis

Usage

delete_chain_masks ***chain_name***... | **-All**

Description

Removes chain masks from the specified scan chains.

The delete_chain_masks command deletes chain masks from the specified scan chains defined with the add_chain_masks command. You can delete the definitions of specific chain name or all chains.

Arguments

- ***chain_name***

  A repeatable string that specifies the names of the chain mask definitions that you want to delete.

- **-All**

  A switch that deletes all chain mask definitions.

Example

The following example defines two chain masks then deletes all chain masks:

```
add_chain_masks chain2 chain3
delete_chain_masks -all
```

Related Commands

add_chain_masks

report_chain_masks

delete_clocks

Context: all contexts

Mode: setup

Usage

delete_clocks ***primary_input_pin***... | **-All**

Description

Removes primary input pins from the clock list.

The delete_clocks command deletes primary input pins from the clock list. If you delete an equivalence pin, the command also deletes all of the equivalent pins from the clock list.

Arguments

- ***primary_input_pin***

  A repeatable string that specifies the primary input pins to delete from the clock list.

- **-All**

  A switch that deletes all pins from the clock list.

Example

The following example deletes an incorrectly added clock from the clock list:

```
add_clocks 1 clock1
add_clocks 1 clock2
delete_clocks clock1
```

Related Commands

add_clocks

report_clocks

delete_config_elements

Context: all contexts including unspecified

Mode: all modes

Usage

delete_config_elements *config_objects* |
{*config_name_list* [-in_wrapper *wrapper_object_spec*]}

Description

Deletes one or more configuration elements. Only elements which can be added can be deleted.

Singular elements such as a wrapper declared with Repeatable set to single in the metadata and properties cannot be deleted as they always exist as soon as their parent wrapper exist.

Arguments

- *config_object*

  An optional collection containing one or more configuration elements. When using a collection of object, the -in_wrapper option cannot be used as the objects contains the full path of the object

- *config_name_list*

  An optional value that specifies the name of a configuration element or a Tcl list containing names of configuration elements. When the -in_wrapper option is specified, the names are relative to the specified wrapper element.

- -in_wrapper *wrapper_object_spec*

  An optional option value pair that is used to specify the parent wrapper in which the configuration elements are to be deleted. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The names in the config_name_list are relative to the specified “wrapper_object_spec”.

Example

The following example deletes elements using the full name of elements, using a relative name with respect to a parent wrapper and using a collection of objects.

```
> delete_config_elements {
    DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(1)
    DftSpecification(mod1,rtl)/IjtagNetwork/HostScanInterface(2) }
> delete_config_elements {HostScanInterface(1) HostScanInterface(2)} \
    -in_wrapper DftSpecification(mod1,rtl)/IjtagNetwork
> delete_config_elements [get_config_elements HostScanInterface \
                            -in DftSpecification(mod1,rtl)/IjtagNetwork
```

Related Commands

add_config_element

get_config_elements

delete_config_messages

Context: all contexts

Mode: all modes

Usage

delete_config_messages ***config_object_spec***
[-hierarchical]
[-all | [-error] [-warning] [-info]]
[-silent]

Description

Deletes error, warning or info message that were added to configuration elements using the add_config_message command.

Arguments

- ***config_object_spec***

  A required value that specifies a configuration object on which to delete the messages. The config_object_spec value is the name of a configuration element or a collection containing a single configuration element object.

- -hierarchical

  An optional Boolean option that specifies that the config message should be deleted on the specified config_object_spec and all elements below it. The -hierarchal option is ignored when the config_object_spec is a property.

- -all

  An optional Boolean option that specifies that all message types are to be deleted. The use of the -all option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages to be deleted are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages to be deleted are of type warning. The use of the -error option is mutually exclusive to the specification of the -error, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -info

  An optional Boolean option that specifies that the messages to be deleted are of type info. The use of the -error option is mutually exclusive to the specification of the -all option. When none of the -error, -warning and -info options is specified, -all is assumed.

- -silent

  An optional Boolean option that disables the warning that is normally generated if the specified configuration element does not contain any messages of the specified type.

Example

The following example deletes all message on and below the wrapper DftSpecification(moda,rtl). The addition and deletion of messages on the DftSpecification and DefaultsSpecification/DftSpecification wrappers is automatically handled by the process_dft_specification command.

```
delete_config_messages DftSpecification(moda,rtl) -hierarchical
```

Related Commands

add_config_message

get_config_messages

report_config_messages

delete_config_tabs

Context: all contexts

Mode: all modes

Usage

delete_config_tabs [*config_object_spec*] [-silent]

Description

Deletes one or many configuration tree tabs from the Configuration Data window that was previously added using add_config_tab or display_specification.

When no config_object_spec value is specified then all existing tabs are deleted.

Arguments

- *config_object_spec*

  An optional value that specifies one or more configuration wrappers. The value is either the name of a wrapper, a tcl list of names of wrappers or a collection containing one or more configuration wrappers. The wrappers are the root wrapper displayed in the tab.

- -silent

  An optional value that disables the warning that is normally generated when config_object_spec contains a configuration element that is not a root wrapper displayed in an existing tab or when no configuration tab exists at all.

Example

The first example deletes the tab in which the DftSpecification(mod1.rtl) wrapper is displayed. The second example deletes all tabs and suppresses the warning that would be generated if there are no tab to delete.

```
> delete_config_tabs DftSpecification(mod1.rtl)
> delete_config_tabs -silent
```

Related Commands

add_config_tab

display_specification

delete_connections

Context: all contexts

Mode: insertion

Usage

delete_connections ***obj_spec*** [-silent]

Description

Disconnects net, pin, or port objects.

If the disconnected object is a pin, the pin is left without a connection in the instance independent of whether the pin is an input, output, or inout pin. If the object is a port, if the associated net of the same name is connected to any other object, the tool creates a uniquified net and uses it in all of the connections the original net made. This results in input and inout ports with no fanout, and output and inout ports with no fanin. If the object is a net that is also a pin, the tool creates a uniquified net and uses it in all of the connections the original net made. This results in input and inout pins with no fanout, and output and inout ports with no fanin.

You must be careful when using the delete_connection command inside a non-uniquified design. You must only do the move or disconnection once per module and once per generate loop count. See how the parent_instance and the leaf_name_hash attributes are used in section 3 of Example 5 in the get_dft_cell command description to perform this robustly.

Arguments

- ***obj_spec***

  The obj_spec must be a single net, pin or port name, a Tcl list of net, pin, or port names, or a collections of net, pin or port names created by the get_nets, get_pins, or get_ports commands.

  If an object within obj_spec does not exist and the -silent option is specified, the object is ignored. If an object within obj_spec does not exist and the -silent option is not specified, the tool generates an error.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Example 1

The following example disconnects the net or constant that may be connected to input pin u1/A.

delete_connections u1/A

```
{}
```

Example 2

The following example disconnects the net or constant that may be connected to the pin within the generated collection.

delete_connections [get_pins ABC*/TE]

```
{}
```

Example 3

The following example disconnects the internal fanout of the input pin ABC/in1. It does that by renaming the net ABC/in1 to ABC/in1_ts1. The pin ABC/in1 ends up with no fanout.

delete_connections [get_nets ABC/in1]

```
{ABC/in1_ts1}
```

Example 4

The following example disconnects the internal fanin of the output port out1. It does that by renaming the net out1 to out1_ts1. The port out1 ends up with no fanin.

delete_connections out1

```
{out_ts1}
```

Related Commands

create_connections
get_nets
get_icl_pins
get_icl_ports
move_connections

delete_core_descriptions

Context: all contexts

Mode: setup

Usage

delete_core_descriptions {**-all** | **-cores** ***core_name_list***} [-modes *mode_name_list*]

Description

Removes core descriptions that have been read in using the read_core_descriptions command from memory.

You cannot delete a core description that is instantiated in another core description unless you specify to delete the parent core description also, by using the -all switch for example.

If a corresponding core instance exists for the core description, this command deletes the core instance along with the core description.

If you specify to delete multiple descriptions and the deletion of one or more core descriptions fails, this command will continue deletion of the remaining core descriptions.

The deletion of a top-level core description is only allowed in a non-extraction flow.

Arguments

- **-all**

  A switch that specifies to delete all core description files in the current design.

- **-cores** ***core_name_list***

  A required switch and Tcl list that specifies the core names of the core descriptions to be deleted.

- -modes *mode_name_list*

  An optional switch that specifies the mode of the core to delete.

Example

The following example reads in the core descriptions from the top_core_des file and then reports the defined core descriptions. The top-level core description is then deleted from memory. The second_level_core becomes the top of the design. The report_core_descriptions command then reports the defined core descriptions after the top-level core description is deleted.

```
set_context patterns -scan_retargeting
read_core_descriptions top_core_des
report_core_descriptions

Name                  Mode Name    Mode Type
-------------------   -----------  -----------
* top                 retargeting  retargeting
  second_level_core   retargeting  retargeting
```

```
  core1              mode_M1      internal
                     mode_M3      internal
  core2              mode_M2      internal
  core3              mode_M2      internal
                     mode_M3      internal

delete_core_descriptions -core top
set_current_design second_level_core
report_core_descriptions

Name                 Mode Name    Mode Type
-------------------  -----------  -----------
* second_level_core  retargeting  retargeting
  core1              mode_M1      internal
                     mode_M3      internal
  core2              mode_M2      internal
  core3              mode_M2      internal
                     mode_M3      internal
```

Related Commands

add_core_instances
delete_core_instances
read_core_descriptions
report_core_instances
report_core_descriptions
write_core_description

delete_core_instances

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Usage

delete_core_instances {**-all** | [-cores *core_names*] | [-modules *module_objects*] | **[**-instances *instance_objects*] | [-current_design]} [-modes *mode_names*] [-silent]

Description

Removes an instance binding created with the add_core_instances command.

You can use this command to retarget patterns for different configurations sequentially in the same tool session. After completing one configuration and before switching to the next configuration, you use the delete_core_instances command to delete the old core instances so that the core instances used in the next session can be added.

Arguments

- **-all**

  A switch that specifies to remove all bindings that have been added with the add_core_instances command.

- -cores *core_names*

  An optional switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be removed.

- -modules *module_objects*

  An optional switch and string pair that specifies an object name, a Tcl list of object names (hierarchical names), or a collection of objects. This command removes the instance bindings for each instantiation of those modules in the design.

- -instances *instance_objects*

  An optional switch and string pair that specifies an object name, a Tcl list of object names (hierarchical names), or a collection of objects. This command removes the instance bindings for each instance in the list.

- -current_design

  An optional switch that specifies to delete the core instance with respect to the current design. This switch is mutually exclusive with the -core, -modules, and -instances switches.

- -modes *mode_names*

  An optional switch and string pair that specifies the list of modes to delete for each specified core instance.

- -silent

  An optional switch that specifies to suppress warning messages.

Example 1

The following example adds the same instance twice. The second attempt fails with a warning, deletes it, and re-adds it successfully.

```
add_core_instances -instances core_1 -cores my_core
//  Added core instance 'core_1'.
add_core_instances -instances core_1 -cores my_core
//  Warning: Instance 'core_1' has already been added. Skipping.
delete_core_instances -instances core_1
add_core_instances -instances core_1 -cores my_core
//  Added core instance 'core_1'.
```

Example 2

The following example adds two instances of the same module and then deletes both of them.

```
add_core_instances -instances {core_1 core_2} -cores my_core
//  Added core instance 'core_1'.
//  Added core instance 'core_2'.
delete_core_instances -instances {core_1 core_2}
```

Example 3

The following example adds two instances and then attempts to delete the two instances as well as one other instance that has not been added. The -silent usage suppresses the warning messages that would be printed for the three non-existing core instances.

```
add_core_instances -instances {core_1 core_2} -cores my_core
delete_core_instances -instances {core_1 core_2 core_3}
//  Warning: Instance 'core_3' of module 'my_core_2' has not been added as
   a core instance. Skipping.
delete_core_instances -instances {core_1 core_2 core_3} -silent
```

Example 4

The following example adds two instances of the same module and then deletes both of them through the module.

```
add_core_instances -instances {core_1 core_2} -cores my_core
//  Added core instance 'core_1'.
//  Added core instance 'core_2'.
delete_core_instances -modules my_module
```

Related Commands

add_core_instances
read_core_descriptions
report_core_instances
write_core_description

delete_design

Context: all contexts

Mode: setup

Usage

delete_design

Description

Removes the current design including the flattened model, and all tool data that is design-specific.

This command also closes any open DFTVisualizer windows. Note that this command does not delete cell libraries unless a flattened model is deleted.

When the delete_design command is deleting a flat model, and that flat model was saved with a list of enabled/disabled commands, then the restriction on the disabled commands is removed. In other words, all commands allowed in the tool will be enabled for the next design.

Arguments

None

Example

The following example removes all current designs:

```
delete_design
```

Related Commands

delete_cell_library
read_verilog
set_current_design
set_design_macros
read_flat_model

delete_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Usage

delete_dfm [-vacuum]

Description

Deletes all the DFM rules from the open VDB.

Arguments

- -vacuum

  An optional switch that specifies to vacuum the VDB to reduce disk size. This operation may take a long time because Tessent Diagnosis reallocates the VDB memory.

Related Commands

import_dfm

delete_display_data

Context: all contexts

Mode: setup, analysis

Usage

delete_display_data
Error_pattern |
{**PATtern_index** ***pattern_index*** [-Internal | -External]} |
{**Parallel_pattern** ***pattern_number***} |
Fault_status |
TIe_value |
COnstrain_value |
Seq_depth_data |
{**CLock_cone** ***pin_name***} |
CAPTURE_PROCedure ***procedure_name*** |
Drc_pattern {{**Test_setup** [{{-Cycle | -Time} *n1*} [*n2* | End]]} | **Load_unload** | **SHIft** |
SKew_load | **SHADOW_Control** | **Master_observe** |
SHADOW_Observe | **STate_stability** | **TEST_End** [*time* | -All]}

Note

The following arguments are not supported in dft -scan: Pattern_index, Parallel_pattern, Fault_status, and Seq_depth_data.

Description

Removes a data column from the DFTVisualizer Data window.

This command has the same arguments and usage as the add_display_data command.

You can control the display of a column by right clicking on the title bar and selecting/deselecting the appropriate option.

Examples

The following example adds a column named “drc test_setup -cycle 5 10” that displays the simulated values for cycles 5 through 10 of the test_setup procedure:

```
add_display_data drc_pattern test_setup -cycle 5 10
```

The following example deletes the column added in the preceding example:

```
delete_display_data drc test_setup -cycle 5 10
```

Note

The argument string you specify when deleting a column must match exactly the column name, including spaces.

Related Commands

add_display_data

delete_display_instances

Context: all contexts

Mode: setup, analysis

Usage

delete_display_instances {***gate_id#*** | ***instance_name***}... | **-All**
[-DIsplay *window_name*]

Description

Removes the specified design instances from a DFTVisualizer display window.

Arguments

- ***gate_id#***

  A repeatable integer that specifies the identification number of a gate you want to remove from the specified display in DFTVisualizer. The value of the gate_id# argument is the unique identification number the tool automatically assigns to every gate in the design during the model flattening process.

- ***instance_name***

  A repeatable string that specifies the name of an instance you want to remove from the specified display in DFTVisualizer.

- **-All**

  A switch that removes all the instances in the specified DFTVisualizer display.

- -Display *window_name*

  An optional switch and literal that species which DFTVisualizer windows to delete instance data from. You can specify multiple windows. Literal options include:

  DEBug — A literal that specifies the Debug window.

  DESign — A literal that specifies the Design window.

  DAta — A literal that specifies the Data window.

  ALl — A literal that specifies all windows.

  The default is to delete instances from the Debug window only.

Examples

The following example first directs DFTVisualizer to display three instances in its Debug window, then removes one of the instances from the display:

```
add_display_instances 32 i_2_16 62
delete_display_instances 62
```

The next example removes the remaining gates from the Debug window, leaving the window blank:

delete_display_instances -all

Related Commands

add_display_data
add_display_instances
open_visualizer
report_display_instances

delete_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

delete_edt_blocks **-All** | ***block_name***

Description

Removes the data for the specified EDT block(s) from the tool's internal database.

The delete_edt_blocks command removes all data for the specified EDT block(s) from the tool's internal database, making the block(s) unavailable for use with the set_current_edt_block command.

If you delete the current EDT block, leaving just one remaining EDT block defined, the tool automatically sets the EDT context to the remaining block. The tool does not reset the EDT context, however, if there is more than one defined EDT block remaining; in this case subsequent context-sensitive commands that define attributes for a given EDT block will return an error until you set the EDT context using the add_edt_blocks or set_current_edt_block command.

You set or change the EDT context when you restrict the applicability of certain commands to a particular EDT block by issuing the add_edt_blocks or set_current_edt_block command. The EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command or change the context to a different EDT block using the add_edt_blocks or set_current_edt_block command. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. Most commands like set_system_mode, report_statistics and create_patterns, are unaffected by the EDT context and always operate on the whole design.

Arguments

- **-All**

  A switch that specifies to remove the data for all EDT blocks from the tool's internal database.

- ***block_name***

  A string that specifies to remove the data for a particular EDT block from the tool's internal database.

Example 1

The following example displays currently defined EDT block names, then removes the definition for the one representing the current EDT block.

report_edt_blocks

```
//  my_core1_block
//  my_core3_block   (current block)
```

delete_edt_blocks my_core3_block
report_edt_blocks

```
//  Block deleted. my_core1_block is now the current EDT block.
```

Example 2

The following example shows how the tool manages the current EDT context when there are more than two blocks defined.

report_edt_blocks

```
// my_core1_block
// my_core2_block
// my_core3_block
// my_core4_block   (current block)
```

delete_edt_blocks my_core3_block
report_edt_blocks

```
// my_core1_block
// my_core2_block
// my_core4_block   (current block)
```

delete_edt_blocks my_core4_block

```
// Block deleted. No EDT block now selected as current block.
```

set_edt_options -channels 4

```
// Error: No current EDT block defined.
```

set_current_edt_block my_core2_block
set_edt_options -channels 4
report_edt_configuration

```
//
// EDT block "my_core2_block"
// --------------------------
// IP version:                     3
// External scan channels:         4
// Bypass logic:                   On
// Lockup cells:                   On
// Clocking:                       edge-sensitive
//
// Note: Only current EDT block "my_core2_block" is reported. Use "report
//    edt configuration -all_blocks" to report the configurations of all
//    blocks.
```

Related Commands

add_edt_blocks
report_edt_blocks
set_current_edt_block

delete_edt_configurations

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the add_edt_configurations command.

Usage

delete_edt_configurations **-All** | ***config_name***

Description

Deletes specified compression configurations from all blocks in a design.

In a multi-block design flow, all subsequently added blocks inherit the defined compression configurations. The compression configuration parameters are then set for each block context.

Arguments

- **-All**

  A required switch that deletes all the compression configurations from all the blocks of the design.

- ***config_name***

  A required string that specifies the name of a single compression configuration to delete from all the blocks of the design.

Example

The following example reports the currently defined compression configurations in a design, deletes one, and reports the remaining compression configurations.

report_edt_configuration

```
// my_core1_block
// IP version:                    1
// Bypass logic:                  On
// Lockup cells:                  On
// Clocking:                      edge-sensitive
// Input channels:   [manufacturing_test : 10] [system_test : 4]
// Output channels:  [manufacturing_test : 5] [system_test : 2]
// Bypass chains: 5
```

delete_edt_configurations system_test
report_edt_configuration

```
// my_core1_block
// IP version:                1
// Bypass logic:              On
// Lockup cells:              On
// Clocking:                  edge-sensitive
// Input channels: [manufacturing_test : 10]
// Output channels:[manufacturing_test : 5]
```

```
// Bypass chains: 5
```

Related Commands

add_edt_configurations
report_edt_configurations
set_current_edt_configuration

delete_edt_connections

Context: dft -edt

Mode: setup

Usage

delete_edt_connections [-Signal ***type*** [-Blocks {*block_name* ...} | -All_blocks]] [{-From *connection_pathname*} {-To {*connection_pathname*...}}]

Description

Deletes connections previously specified with the add_edt_connections command.

Arguments

- -Signal ***type***

  Optional switch and literal pair that specifies an EDT control signal. Valid literals include:

  clock — specifies the edt_clock pin.

  reset — specifies the edt_reset pin.

  scan_enable — specifies the top-level scan_en pin.

  bypass — specifies the edt_bypass pin.

  update — specifies the edt_update pin.

  edt_configuration — specifies the edt_configuration pin for dual configuration applications.

  master_clock — specifies the master_clock pin. Use in latch-based EDT logic for LSSD designs only.

  slave_clock — specifies the slave_clock pin. Use in latch-based EDT logic for LSSD designs only.

  input_channel *N* — appended with the channel number to specify an EDT input channel.

  output_channel *N* — appended with the channel number to specify an EDT output channel.

- -Blocks *block_name* **...**

  Optional switch and repeatable string pair that specifies one or more blocks to target. If no blocks are specified, the current one is assumed.

- -All_blocks

  Optional switch that specifies all blocks. If no blocks are specified, the current one is assumed.

- -To *connection_name*

  Optional switch and repeatable string pair that specifies the pathname of the destination pin/node for the targeted connection.

- -From *connection_name*

  Optional switch and string pair that specifies the pin pathname of the source pin/node for the targeted connection.

Example

The following example adds block-level EDT signal connections and then deletes one and re-adds it.

Specify connections to block-level EDT signals:

```
add_edt_connections -signal input_channel 1 -to TK_channel_in[0] -block B1
add_edt_connections -signal output_channel 1 -to TK_channel_out[0] -block B1
add_edt_connections -signal input 1 -to TK_channel_in[1] -block B2
add_edt_connections -signal output 1 -to TK_channel_out[1] -block B2
add_edt_connections -signal clock -to TK_clk -blocks B1 B2
add_edt_connections -signal update -to TK_upd -all_blocks
add_edt_connections -signal bypass -to /tap_i/tap_ctrl_i/edt_bypass_inst -all_blocks
```

Delete an EDT connection:

```
delete_edt_connections -signal bypass -all_blocks
```

Re-add the EDT connection:

```
add_edt_connections -signal bypass -to /tap_i/tap_ctrl_i/edt_bypass_inst -all_blocks
```

Related Commands

add_edt_configurations

report_edt_connections

delete_false_paths

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

Mode: setup, analysis

Usage

delete_false_paths [-All] [-Verbose]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]

Description

Removes definitions of false paths.

The delete_false_paths command deletes from the tool's internal database, false paths you defined using the add_false_paths command.

Arguments

Tip: When using the following arguments, you can include any number of asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that specifies to delete all previously defined false paths.

- -Verbose

  An optional switch that directs the tool to display the definition of each deleted false path as it is deleted.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified source node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with

this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified sink node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

Note

For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “delete_false_paths... -to CLK2” would delete false paths that have an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified false path. For example, the following false path specification:

  delete false paths -from A -through B C -through D E -to F

  specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, the tool will

consider all paths from the specified starting points to the specified end points to be false paths.

If a through node does not translate to an internal gate, the tool will transcript an error message and not delete the false path definition.

Example

The following example displays current false path definitions, then deletes one of them, echoing its definition as it is deleted:

report false paths -all

```
// False Path -from /my_design/a /my_design/b -through /my_design/u1
// False Path -to /my_design/c
// False Path -through /my_design/u24 /my_design/u25
// Total reported paths = 3
```

delete false paths -verbose -to /my_design/c

```
// Deleting false paths ...
// False Path -to /my_design/c ( /my_design/c/Y (36) )
// Deleted 1 false paths.
```

Refer to the add_false_paths command for examples of signal matches for pathnames specified using the asterisk (*) character.

Related Commands

add_false_paths
delete_multicycle_paths
read_sdc
report_false_paths
report_multicycle_paths

delete_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Bridge Faults usage:
delete_fault_sites {**-All** | {***bridge_name***… **-NAME**}}

Path Delay Faults Usage:
delete_fault_sites {**-All** | **-Unsensitizable_paths** | ***path_name***…}

UDFM Faults Usage:
delete_fault_sites {**-All** | ***file_name***}

Description

Removes bridge entries, delay paths, user-defined fault sites, and any associated faults from the current fault list. For UDFMs, all faults are removed.

Depending on the current fault model, deletes specified fault sites (bridge entries or delay paths) from the current fault list. The delete_fault_sites command removes faults loaded with the read_fault_sites command. When a fault site is removed from the current fault list, the associated patterns are also removed from the internal pattern set.

For more information on bridge faults and the bridge fault definition file, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information on path delay faults and the path definition file, refer to "Creating a Path Delay Test Set" in the *Tessent Scan and ATPG User's Manual*.

For more information on UDFMs, see "User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-All**

  A switch that specifies to remove all fault sites from the current fault list and deletes all faults associated with the deleted fault sites.

- ***bridge_name***

  A repeatable string that specifies a bridge name.

- **-NAME**

  A switch that specifies the argument strings preceding the switch are bridge names. This switch is required just once, whether you enter one or multiple *bridge_name* strings.

- **-Unsensitizable_paths**

  A switch that directs the tool to perform an ATPG analysis of each path, and to delete those proved unsensitizable. For a list of critical paths identified by static timing analysis, for example, use this switch to identify and delete the unsensitizable paths.

Note

This ATPG process may be very lengthy. A progress message is displayed after every 100 paths analyzed, as well as at the end of the analysis.

- ***path_name***

  A repeatable string that specifies the name of an existing path that resides in the current path definition file and whose path delay faults you want to remove from the current path delay fault list.

- **file_name**

  A required string that specifies a UDFM file name. Use this option to delete all fault sites associated with a specified UDFM file.

Example 1

The following example reads the bridge information in the bridge fault definition file, */user/design/bridgefile*, then deletes two of the three valid entries added to the fault site list:

set_fault_type bridge
read_fault_sites /user/design/bridgefile

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

```
BRIDGE "U7.BRIDGE" {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
BRIDGE "U5.BRIDGE" {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
BRIDGE "U0.BRIDGE" {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
```

delete_fault_sites U5.BRIDGE U7.BRIDGE -name
report_fault_sites

```
BRIDGE "U0.BRIDGE" {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
}
```

Example 2

The following path delay example reads the path information from the file, */user/design/pathfile*, and deletes one of the two paths:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH “path0” =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
PATH “path1” =
    PIN /I$6/Q + ;
    PIN /I$35/B1 + ;
    PIN /I$35/C1 + ;
    PIN /I$1/I$649/IN + ;
    PIN /I$1/I$649/OUT - ;
    PIN /I$5/D - ;
END ;
```

delete_fault_sites path0
report_fault_sites

```
PATH “path1” =
    PIN /I$6/Q + ;
    PIN /I$35/B1 + ;
    PIN /I$35/C1 + ;
    PIN /I$1/I$649/IN + ;
    PIN /I$1/I$649/OUT - ;
    PIN /I$5/D - ;
END ;
```

Example 3

The following example deletes all fault sites associated with the *c090_std.udfm* UDFM model:

delete_fault_sites /home/design/c090_std/c090_std.udfm

Example 4

The following example deletes all fault sites from the current fault list:

delete_fault_sites -All

Related Commands

add_fault_sites
add_faults
analyze_fault
delete_faults
read_fault_sites
report_fault_sites
set_fault_type
write_fault_sites

delete_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

delete_faults ***fault_model_specific_options*** [*power_aware_options*]
[{> | >>} *file_pathname*]

Path Delay Faults Usage:
delete_faults {**-All** | ***name_of_delay_path***… | **-Class** {***class_name*** | ***class_code***}…}
[-Both | -Rise | -Fall] [-UNTestable]

Stuck/Transition/Toggle/Iddq Faults Usage:
delete_faults { { { {-**All** [-Keep_protected]} | **-Protected** |
{**-Class** {***class_name*** | ***class_code***}…} |
{***object_expression***… [-PIN | -INstance | -MODule [-PIN *module_pin_pathname* ...]]} |
{**-CLOCK_Domains** {**ALL** | ***clock_pathname…***} [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]} |
{**-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}}}[-Stuck_at [01 | 0 | 1]]} |
{[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}} [-VERbose]

Bridge Faults Usage:
delete_faults {**-All** | {**-Class** {***class_name*** | ***class_code***}…}
| {{***bridge_name***… -NAME} | {***net_pathname***… -SINGle}
| {***instance_name***… -SINGle}}
| {**-CLOCK_Domains** {**ALL** | ***clock_pathname…***} [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]}
| {**-CAPture_procedures** {**ALL** | ***capture_proc_name…***}}
| {[-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls]}

UDFM Fault Usage:
delete_faults { **-All** | { **-Class** { ***class_name*** | ***class_code*** } ... }
| { ***object_expression*** ... [-INstance] }
| { **-CLOCK_Domains** { **ALL** | **clock_pathname** ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains] }
| [-CLOCK_Cones]
| { [**-Udfm_type** ***name***] [**-CEll** ***name***] [**-MODule** ***name***]
[**-FAult** ***name***] } [-VERBose]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):
[[–ON_domains] | [-OFf_domains] | [-POWer_domains {*domain_name* …}]]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

Removes faults from the current fault list.

The delete_faults command deletes from the fault list, faults you added using the add_faults or read_faults command. The power-aware options remove faults based on power domains and/or power features after you have loaded a CPF or UPF file.

Alternatively, you can also delete a large number of *non-protected faults* using the read_faults commands's -Delete switch.

When you issue this command, the tool discards all patterns in the current test pattern set. To save the current test patterns you must explicitly save them with the write_patterns command prior to issuing the delete_faults command.

By default, the delete_faults command ignores the constraint values implied in the data path when deleting faults in the data path. However, the tool still considers the constraint values for clock path faults.

You cannot use a single command to delete_faults for clock domains, capture procedures, and specific classes, objects, or paths. Use a separate command instance for each.

When bridge faults are deleted, the currently loaded faults sites remain intact.

Arguments

- **-All**

  A required switch that deletes all faults in the current fault list.

- -Keep_protected

  An optional switch, specified with the -All switch, that deletes all faults except the protected faults.

- ***name_of_delay_path***

  A required, repeatable string that specifies the name of the delay path to delete from the current fault list. The name must correspond to the name of a delay path defined in the path definition file.

- **-Protected**

  A required switch that deletes only the protected faults.

- **-Class** {***class_name*** | ***class_code***}…

  A required switch and repeatable literal pair that deletes one or more classes of faults. This argument is valid only for unprotected faults.

  Table 7-8 lists the valid fault *class_codes* and their associated fault *class_names.*

Table 7-8. Fault Class Codes and Class Names

Fault Class Codes	Fault Class Names	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR

Table 7-8. Fault Class Codes and Class Names (cont.)

Fault Class Codes	Fault Class Names	Fault Class Coverage
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- -Both | -Rise | -Fall

  An optional switch that specifies which faults to delete for each path already added via the Add Paths command. These switches are valid only for path delay faults.

  -Both — Deletes both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise — Deletes only the slow-to-rise faults.

  -Fall — Deletes only the slow-to-fall faults.

- ***object_expression***

  A required string representing a list of instances or pins associated with the faults you want to delete from the current fault list. Note that in the case of UDFM, you can specify only instances. You can use regular expressions that include any number of asterisk (*) and question mark (?) wildcard characters.

  Instance pathnames must be ATPG library cell instances. Pin pathnames must be ATPG library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of an ATPG library model, it is ignored. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -PIN

  An optional switch that uses the *object_expression* to match only pin pathnames and then, delete the associated faults.

- -INstance

  An optional switch that uses the *object_expression* to match only instances and then, delete the associated faults.

- -MODule [-PIN *module_pin_pathname* ...] (Stuck/Transition/Toggle/Iddq only)

  A switch that specifies to interpret the object_expression argument as a module pathname. All instances of the module are affected. You can use the asterisk (*) and question mark (?) wildcards for the object_expression argument, and the tool adds the fault for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given object_expression and deletes faults for every pin specified in module_pin_pathname list.

- -Untestable

  An optional switch that deletes all identified, untestable path delay faults. Untestable faults are common when using random patterns. This includes faults that the tool cannot detect due either to constraints or the use of a single capture clock.

- ***bridge_name***

  A required, repeatable string that specifies the name of a bridge. The faults associated with the specified bridge will be deleted from the current fault list.

- ***net_pathname***

  A required, repeatable string that specifies the name of a single net. The faults associated with the specified net will be deleted from the current fault list.

- ***instance_name***

  A required, repeatable string that specifies the name of an instance. The faults associated with the specified instance will be deleted from the current fault list.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  Optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

  If the string specifies a net, each bridge entry associated with the net is deleted.

  If the string specifies an instance that contains a non-dominant net included in a bridge entry, the associated faults are deleted.

- **-CLOCK_Domains** {**ALL** | ***clock_pathname…***}

  A required switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to delete_faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock. The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

  clock_pathname — A repeatable string that specifies a particular clock.

  Be aware that a fault potentially detectable by a clock you specify with this switch will be deleted even if it might also be detectable by an unspecified clock.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from deleting faults in equivalent clock domains.

- -INCLUDE_INTER_NORACE_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to take user-defined non-race clocks into account when deleting faults in the clock domain. You can use this switch in non-setup mode and when the clock restriction is set to be domain_clock.

- **-CAPture_procedures** {**ALL** | ***capture_procedure_name…***}

  A required switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to delete_faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  See the set_capture_procedures command description for information about enabling or disabling named capture procedures.

- -Stuck_at 01 | 1 | 0

  An optional switch and literal pair that specifies the stuck-at or transition faults to delete from the fault list.

  01 — A literal specifying that for stuck-at faults the tool delete both stuck-at-0 and stuck-at-1 faults; or for transition faults the tool delete both slow-to-rise and slow-to-fall faults. This is the default.

  0 — A literal specifying to delete only the stuck-at-0 faults (slow-to-rise faults for transition faults).

  1 — A literal specifying to delete only the stuck-at-1 faults (slow-to-fall faults for transition faults).

The following switches, which support stuck-at, transition, toggle, Iddq and bridge faults, delete faults at certain locations that are ATPG untestable in functional mode or should not be tested (to avoid over-testing). For bridge faults, the switches apply to any bridge that has at least one end in a specified location. When justified by your knowledge of the design, use the switches to fine-tune the fault population and improve the quality of the tool's coverage statistics.

- -SCAN_Enable

  An optional switch that deletes faults that only fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch deletes all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that deletes all faults that only fan out to clock ports, Set/Reset ports or Read/Write control ports of memory elements, including RAM and ROM. This switch considers any memory elements, not just scan cells.

- -IO

  An optional switch that deletes faults that are either only controlled by PIs or only observed by POs. This switch only applies to PIs that are either not defined as a clock or defined as clock (or read/write control), but constrained to be off during capture.

- -ASYnchronous_controls

  An optional switch that deletes all faults that only fan out to Set/Reset ports of state elements and RAMs. Notice that this switch applies to a subset of the faults the -CLOCK_Cones switch deletes.

- -VERbose

  An optional switch that outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

  ```
  // Note:  Adding faults for 330 fault sites.
  ```

  If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

- **-Udfm_Type *name***

  A required switch and string pair that specifies a type of UDFM fault to delete from the current fault list. The name can contain wildcards.

- **-CEll *name***

  A required switch and string pair that specifies a library cell. Use this option to delete all faults associated with a specified cell. The name can contain wildcards.

- **-MODule *name*** (UDFM only)

  A required switch and string pair that specifies a module. Use this option to delete all faults associated with a specified module. The name can contain wildcards.

- **-Class *class_type***

  A required switch and repeatable literal pair that deletes the internal fault entries that belong to the class specified by the *class_type* argument. The *class_type* argument can be either the fault class code or a fault class name. For more information on fault classes, see "Fault Classes" in the *Tessent Scan and ATPG User's Manual*.

- **-FAult *name***

  A required switch and string pair that specifies a UDFM fault name. The name can contain wildcards.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that removes faults on all power-on domains.

- -OFF_domains

  An optional switch that removes faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that removes faults on the specified power domains (*domain_name*).

- -Isolation_cells

  An optional switch that removes faults on all isolation cells.

- -Level_shifters

  An optional switch that removes faults on all level shifters.

- -REtention_cells

  An optional switch that removes faults on all retention cells.

Example 1

The following example deletes a stuck-at-0 fault from the current fault list after adding all the faults to the circuit, but before creating patterns:

set_system_mode analysis
add_faults -all
delete_faults i_1006/i1 -stuck_at 0
create_patterns

Example 2

The following example deletes all stuck-at-0 AU faults. Notice the reduction in AU faults in the statistics report and the corresponding increase in the reported coverages as a result of removing these faults from the calculation. (For information on the formulas used to calculate coverage, refer to "Testability Calculations" in the *Tessent Scan and ATPG User's Manual*.)

report_statistics

```
            Statistics report
-------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                    296       472
-----------------------   -------   -------
UC (uncontrolled)            105       185
DI (det_implication)         127       221
UU (unused)                   18        18
AU (atpg_untestable)          46        48
-----------------------   -------   -------
test_coverage              45.68%    48.68%
fault_coverage             42.91%    46.82%
atpg_effectiveness         64.53%    60.81%
-------------------------------------------
#test_patterns                            0
#simulated_patterns                       0
CPU_time (secs)                         0.1
-------------------------------------------
```

delete_faults -class au -stuck_at 0

```
//  Note: 23 collapsed faults were deleted.
```

report_statistics

```
            Statistics report
-------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                    273       448
-----------------------   -------   -------
UC (uncontrolled)            105       185
DI (det_implication)         127       221
UU (unused)                   18        18
AU (atpg_untestable)          23        24
-----------------------   -------   -------
test_coverage              49.80%    51.40%
fault_coverage             46.52%    49.33%
atpg_effectiveness         61.54%    58.71%
-------------------------------------------
#test_patterns                            0
```

```
#simulated_patterns                      0
CPU_time (secs)                        0.1
-------------------------------------------
```

Example 3

After adding all of the faults in the design, this example removes only the faults from the power domain named PD1:

add_faults -all
delete_faults -power_domain PD1

Example 4

The following example deletes all UDFM type *intra_cell_bridges* associated with all library cells that begin with *mux*:

delete_faults -udfm_type intra_cell_bridges -cell mux*

Related Commands

add_faults
delete_nofaults
read_fault_sites
read_faults
report_fault_sites
report_faults
report_testability_data
set_fault_mode
set_fault_protection
set_fault_sampling
set_fault_type
write_fault_sites
write_faults
write_patterns

delete_flat_model

Context: all contexts

Mode: setup

Usage

delete_flat_model

Description

Deletes a flat model that was previously created either by going to system mode analysis or by using the create_flat_model command.

You typically do not need this command because any command you use that invalidates the flat model automatically deletes the flat model at the same time. The tool also automatically deletes the flat model when you enter insertion mode.

In addition, the tool deletes the flat model when you set an attribute on a design module, instance, port, or pin that has the preserve boundary option enabled. Refer to the set_attribute_options -preserve_boundary_in_flat_model option for more information.

To free up used memory, you can use this command to delete the flat model while still in setup mode if you do not need it anymore.

You cannot use this command to delete a flat model that was read in with the read_flat_model command. In this case, you must use the delete_design command to delete the flat model.

Return value

None

Related Commands

create_flat_model

read_flat_model

delete_design

set_attribute_options

delete_icl_modules

Context: all contexts

Mode: setup

Usage

delete_icl_modules {object_spec | -all} [-silent]

Description

Deletes the specified ICL modules from memory.

If any of the specified ICL modules are instantiated below the current design, the ICL elaboration tree is destroyed and you must recreate it using a new set_current_design command.

Arguments

- object_spec

  A value that specifies one or more ICL modules to delete. The value is a single ICL module name, a Tcl list of ICL module names, or a collection of ICL module objects as returned by the get_icl_modules command. When this argument is omitted, all ICL modules are deleted.

- -all

  A switch that specifies to delete all ICL modules in the system.

- -silent

  An optional argument that specifies to not generate error messages when an ICL module specified by object_spec does not exist.

Return value

None

Example

The first example deletes three ICL modules. The second example deletes all ICL modules. The third example deletes all ICL modules with a name starting with a, b or c.

```
delete_icl_modules {mychip tap tdr}
delete_icl_modules -all
delete_icl_modules [get_icl_modules {[abc].*} -regexp]
```

Related Commands

get_icl_modules

read_icl

delete_icl_ports

Context: patterns -ijtag

Mode: setup

Usage

delete_icl_ports ***port_objects*** [-silent]

Description

Undoes the effect of the add_icl_ports command on a specified list of top level ports.

Arguments

- ***port_objects***

  A required string that specifies a design port name, a tcl list of design port names or a collection of design or icl ports.

- -silent

  An optional switch that specifies to suppress the warning messages that are normally generated when the specified port names do not exist or were never the target of previous add_icl_ports commands.

Example

The following example declares then deletes ICL DataInPort D1, and ICL DataOutPorts O1 O2 and O3.

```
set_current_design top
add_icl_ports D1 -type data_in
add_icl_ports {O1 O2 O3} -type data_out
delete_icl_ports {O1 O2 O3}
```

Related Topics

add_icl_ports

delete_icl_scan_interfaces

Context: patterns -ijtag, dft

Mode: setup, insertion (dft context only)

Usage

delete_icl_scan_interfaces ***name_list* | -All**

Description

Deletes the specified scan interfaces previously added by add_icl_scan_interfaces from the new ICL top module that will be created during ICL extraction.

You must set the current design with set_current_design before using the delete_icl_scan_interfaces command.

Arguments

- ***name_list***

  A list that specifies the names of scan interfaces that ICL extraction will delete from the new top ICL module.

- -All

  A switch that specifies that all scan interfaces will be deleted from the new ICL top module.

Example

The following example will delete the ScanInterfaces named TAP and Internal from the new ICL top module during ICL extraction:

```
delete_icl_scan_interfaces {TAP Internal}
```

Related Commands

add_icl_scan_interfaces
get_icl_scan_interface_list
get_icl_scan_interface_port_list
set_icl_scan_interface_ports

delete_iddq_exceptions

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

Mode: setup, analysis

Usage

delete_iddq_exceptions {**-All** | ***original_string…***}

Description

Deletes exceptions to Iddq restrictions added with the add_iddq_exceptions command.

For more information on IDDQ, see "Creating an IDDQ Test Set" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **-All** | ***original_string…***

  Deletes exceptions to Iddq restrictions. Choose from the following choices:

  -All — A required switch that specifies the deletion of all Iddq restrictions.

  original_string — A required repeatable string that identifies a site to which you previously applied Iddq exception. The *original string* is reported by the report_iddq_exceptions command and must be one of the following:

  - A string originally input to an add_iddq_exceptions command
  - A string generated automatically by the analyze_restrictions -Int_float_auto_except command.

Example

The following example removes an Iddq exception:

```
delete_iddq_exceptions /cpu/alu/U2392/
```

Related Commands

analyze_restrictions

set_iddq_checks

delete_ijtag_logical_connection

Context: patterns -ijtag

Mode: setup

Usage

delete_ijtag_logical_connection [-from *source_pin_or_port_name*]
[-to *destination_pin_or_port_name*]

Description

Removes logical connections previously added to a design. You must specify either the source pin/port or the destination pin/port or both source and destination pins/ports.

You must specify the source pin/port or the destination pin/port or both with this command. If both the source and destination pins/ports are specified, the command removes the logical connection between them. If only the source pin/port is specified, then all logical connections that have that pin/port as the source will be removed. If only the destination pin/port is specified, then all logical connections that have that pin/port as the destination will be removed.

Arguments

- **-from** *source_pin_or_port_name*

  An optional switch and value pair that specifies the source pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

- **-to** *destination_pin_or_port_name*

  An optional switch and value pair that specifies the destination pin/port for the logical connection, which should be a top level module port or a hierarchical instance pin in the current top design module.

Example

The following example will add three logical connections in the current design, then delete one logical connection.

add_ijtag_logical_connection -from /i2/in -to /i2/out

add_ijtag_logical_connection -from /i3/in -to /i3/out

add_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination
//  ===========  ==============
//  /i2/in       /i2/out
```

```
//  /i3/in       /i3/out
//  /i4/in       /i4/out
```

delete_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination
//  ===========  ==============
//  /i2/in       /i2/out
//  /i3/in       /i3/out
```

Related Commands

add_ijtag_logical_connection

report_ijtag_logical_connections

delete_input_constraints

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Usage

Context: dft -scan and dft-test_points

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent *equivalent_pin…*] [***constraint_switch***]

Context: patterns -scan

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent *equivalent_pin…*] [***constraint_switch***] [-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]]

Context: patterns -ijtag

delete_input_constraints
[-ALl | *primary_input_pin_name…* | -BIDI_Only | -BIDI_Exclude| -EQuivalent *equivalent_pin…*] [***constraint_switch***]

Description

Removes previously applied constraints from primary input pins.

The delete_input_constraints command removes constraints you previously applied to primary input pins using the add_input_constraints command.

Arguments

- -ALl

  An optional switch that specifies to remove all previously applied input constraints except pin equivalents. This is the default. To remove pin equivalents, use the -Equivalent switch.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin from which to remove the constraint.

- -BIDI_Only

  An optional switch that specifies to remove previously applied constraints only from bidirectional pins.

- -BIDI_Exclude

  An optional switch that specifies to remove previously applied constraints only from primary input pins that are *not* bidirectional.

- -SCAN_INputs

  An optional switch that specifies to remove previously applied constraints only from scan input pins.

- -EQuivalent *equivalent_pin*

  An optional switch and repeatable string that specify the name of a primary input pin from which to remove a previously defined equivalence specification.

- *constraint_switch*

  An optional switch that specifies a particular constraint to remove. The choices, from which you can specify only one, are as follows:

  -C0 — Constant 0

  -C1 — Constant 1

  -CX — Constant X (unknown)

  -CZ — Constant Z (high-impedance)

  -CR0 — Constant that returns to 0

  -CR1 — Constant that returns to 1

  -CT0 — Constant TIE0

  -CT1 — Constant TIE1

  -CTZ — Constant TIEZ

Tip: For additional information about the constraints associated with the preceding switches, refer to the same switches under the add_input_constraints command.

  -C_All — All C-type constraints

  -C_Default — All C-type constraints added using any of the following switches with the add_input_constraints command: -All, -Bidi_only, -Bidi_exclude, -Scan_inputs. This does not include C-type constraints added by specifying the primary input pin name explicitly.

  -C_Specified — C-type constraints added by specifying the primary input pin name explicitly.

- -HOld

  An optional switch that specifies to remove the “hold” restriction, if applicable, from the specified pins. See the description of the -Hold argument to the add_input_constraints command for more information.

- -NO_Z

  An optional switch that specifies to remove the “no-Z” restriction, if applicable, from the specified pins. See the description of the -No_z argument to the add_input_constraints command for more information.

- -SLow_pad

  An optional switch that specifies to remove the “slow-pad” restriction, if applicable, from the specified pins. See the description of the -Slow_pad argument to the add_input_constraints command for more information.

- -CELL *model_name*

  An optional switch and string pair that, together with the -Slow_pad switch, specifies to prefix the instance name of each instance of the ATPG library model of type *model_name* to the *primary_input_pin_name*, and look up each resulting name as the pin directly driven by the primary input.

Example

The following Tessent FastScan example deletes constraints from two primary inputs.

```
delete_input_constraints indata2 -c1
delete_input_constraints indata4 -c0
```

Related Commands

add_input_constraints
report_input_constraints

delete_instances

Context: all contexts

Mode: insertion

Usage

delete_instances ***obj_spec*** [-silent]

Description

Deletes the specified instance objects from the current design.

The ***obj_spec*** can be a single hierarchical instance name, a Tcl list of hierarchical instance names, or a collection of instances.

Arguments

- ***obj_spec***

  A required value that specifies one or more instances to delete.The value is a single hierarchical instance name, a Tcl list of hierarchical instance names, or a collection of instances.

- -silent

  An optional argument that specifies to not generate an error messages when an instance in the obj_spec list does not exist.

Example 1

The following example deletes the two specified instances in the tcl list:

```
delete_instances {u1/u2 u1/u4}
```

Example 2

The following example deletes the collection of instances returned by the get_instances command:

```
delete_instances [get_instances ABC* -hierarchical -of_module XYZ]
```

Related Commands

create_instance
get_common_parent_instance
rename_instance
replace_instances
uniquify_instances

delete_iproc

Context: all contexts

Mode: all modes

Usage

delete_iproc **iproc_list** [-of_module *module_name*] [-silent]

Description

Deletes the specified list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

All iProcs attached to the ICL module are deleted when iproc_list is not specified.

Arguments

- **iproc_list**

  A Tcl list of iProc names.

- -of_module *module_name*

  A required switch and value pair that constrains the command to delete only instances of iProcs attached to the module specified by *module_name*.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProcs or module do not exist.

Return Value

None

Example

The following example deletes the test1 iProc from module mychip and then searches and deletes any remaining iProcs on mychip that begin with “p”.

```
delete_iproc test1 -of_module mychip
detete_iproc [lsearch -inline -all [get_iproc_list p*] -of_module mychip]
```

Related Commands

get_iproc_argument_default
get_iproc_argument_list
get_iproc_body
get_iproc_list
iProc
iProcsForModule

delete_layout_verification

Context: patterns -scan_diagnosis

Mode: analysis

Usage

delete_layout_verification {-List | -Design ***design_index_in_list***}

Description

Reports or deletes stored layout verification information from the current layout database.

Arguments

- **-Design *design_index_in_list***

  A required switch and integer pair that specifies the layout verification data the tool deletes. Use the -list switch to print index numbers for all stored design layout verification information.

- **-List**

  A required switch that directs the tool to list all the designs and corresponding layout rule violation information stored in the layout database. Each design is listed with its assigned Index number — the Index number is used as an argument to the -design switch. The -list switch does not delete any layout verification information, use the -design switch to delete layout verification data from the database.

Example

In the following example, the diagnosis lists the layout verification and design information stored in the open_layout database:

delete_layout_verification -list

```
//  Note: Layout verification information corresponding to the following
//        designs can be deleted from the layout database
//
//  Index    Stored Violation Count  Flat model file name
//  1.       611                     my_design.flat
//  2.       643                     other_design_version.flat
//
//   Note: The current design is my_design.flat (Index 1.)
//   Note: To delete any of the above, please use
//         delete_layout_verification -design <design index in the list>
//
```

In the following example, the diagnosis deletes the layout verification information for design index 1 (my_design.flat):

delete_layout_verification -design 1

```
//  Note: Deleting layout verification information corresponding
//        to my_design.flat.
//  Warning: Since the layout verification information for the
//           current design has been deleted, the layout will be closed.
```

Related Commands

create_layout

delete_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

delete_lfsr_connections ***primary_pin***… | **-All**

Description

Removes connections between the specified primary pins and Linear Feedback Shift Registers (LFSRs).

The delete_lfsr_connections command deletes the connections between the LFSRs and the primary pins specified with the add_lfsr_connections command. You can use the report_lfsr_connections command to display all the current connections between LFSRs and primary pins.

Arguments

- ***primary_pin***

  A repeatable string that lists the primary pins whose connections to LFSRs you want to delete.

- **-All**

  A switch that deletes all of the connections between LFSRs and primary pins.

Example

The following example changes the definition of an LFSR connection by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_connections scan_in.1 lfsr1 2
delete_lfsr_connections scan_in.1
add_lfsr_connections scan_in.2 lfsr1 2
```

Related Commands

add_lfsr_connections

report_lfsr_connections

delete_lfsr_taps

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

delete_lfsr_taps ***lfsr_name*** {***tap_position***… | **-All**}

Description

Removes the tap positions from a Linear Feedback Shift Register (LFSR).

The delete_lfsr_taps command deletes the specified LFSR tap positions added with the add_lfsr_taps command. To display the current tap positions of all defined LFSRs, use the report_lfsrs command.

Arguments

- ***lfsr_name***

  A string that specifies the reference name of the LFSR whose tap positions you want to delete.

- ***tap_position***

  A repeatable string that specifies the list of tap positions that you want to delete from the *lfsr_name*.

- **-All**

  A switch that deletes all of the tap positions from the *lfsr_name*.

Example

The following example changes an LFSR tap position by deleting it and then adding a new tap position:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 Prpg 5 13 -serial -in
add_lfsr_taps lfsr1 2 3 4
add_lfsr_taps lfsr2 1 3
delete_lfsr_taps lfsr1 3
add_lfsr_taps lfsr1 1
```

Related Commands

add_lfsr_taps
set_lfsrs
report_lfsrs

delete_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

delete_lfsrs ***lfsr_name***… | **-All**

Description

Removes the specified Linear Feedback Shift Registers (LFSRs).

The delete_lfsrs command deletes LFSRs defined with the add_lfsrs command. You can use the report_lfsrs command to display a list of the current LFSRs with their current values and tap positions. When you delete an LFSR, the tool also deletes all its taps and pin connections.

Arguments

- ***lfsr_name***

  A repeatable string that specifies the reference names of the LFSRs that you want to remove.

- **-All**

  A switch that deletes all defined LFSRs.

Example

The following example changes the definition of an LFSR by deleting it and then re-adding it with a new definition:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs lfsr3 prpg 5 11 -parallel -out
delete_lfsrs lfsr3
add_lfsrs lfsr3 prpg 5 11 -parallel -in
```

Related Commands

add_lfsrs
set_lfsrs
report_lfsrs

delete_lists

Context: dft -edt, patterns -scan

Mode: analysis

Usage

delete_lists ***pin_pathname***... | **-All**

Description

Removes pins from the pin list the tool monitors and reports on during simulation.

The delete_lists command removes from the monitored pin list, pins that you previously added to the list using the add_lists command. To review the current list of pins, use the report_lists command. To put additional pins on the list, use the add_lists command.

Arguments

- ***pin_pathname***

  A repeatable string that specifies the pins you want to remove from the monitored pin list.

- **-All**

  A switch that removes all currently listed pins from the monitored pin list.

Example

The following example removes an extra added output pin from the monitored pin list:

add_lists /i_1006/o /i_1007/o /i_1008/o
report_lists

```
3 pins are currently monitored.
/i_1006/o
/i_1007/o
/i_1008/o
```

delete_lists /i_1007/o
report_lists

```
2 pins are currently monitored.
/i_1006/o
/i_1008/o
```

Related Commands

add_lists
report_lists
set_list_file

delete_misrs

Context: pattern -scan

Mode: setup, analysis

Usage

delete_misrs {**-all** | ***misr_name***}

Description

Deletes MISRs you specify with the add_misrs command.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- **-all**

  A required switch that specifies removing all MISRs.

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

Example

This example deletes a previously added MISR called misr_0, and then re-adds the MISR as blk1_misr.

add_misrs misr_0 -size 24 -seed 000000 -taps 21 22 24

delete_misrs misr_0

add_misrs blk1_misr -size 24 -seed 0000 -taps 21 22 24

report_misrs -all

```
list of MISRs
blk1_misr  MISR  length=24  in_stream_taps=20,21,23
blk1_misr_pre_unload_value = 000000000000000000000000
```

Related Commands

add_misrs

report_misrs

delete_multicycle_paths

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

Mode: setup, analysis

Usage

delete_multicycle_paths [-All] [-Verbose]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]

Description

Removes definitions of multicycle paths previously read in using the read_sdc command.

The delete_multicycle_paths command deletes from the tool's internal database, multicycle path definitions previously read in by the read_sdc command.

Arguments

Tip: When using the following arguments, you can include asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that specifies to delete all multicycle path definitions.

- -Verbose

  An optional switch that directs the tool to display the definition of each deleted multicycle path as it is deleted.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the output pins of the instance to be starting points. For a clock PI, the tool considers the outputs of all the non-transparent latches, flip-flops and RAMs associated with the clock PI to be starting points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified source node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify

with this argument to be the source of a clock signal and considers the outputs of all non-transparent latches, flip-flops and RAMs associated with this clock signal to be starting points.

If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, the tool considers all the input pins of the instance to be end points. For a clock PI, the tool considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with the clock PI to be end points. The tool considers a pin pathname to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and handles the internal clock pin the same as a clock PI.

  If a specified sink node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

Note

For two port latches with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “delete_multicycle_paths... -to CLK2” would delete_multicycle_paths that have an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path by its associated clock pin pathname. The tool interprets any pin pathname you specify with this argument to be the source of an internal clock signal and considers the data inputs of all non-transparent latches, flip-flops and RAMs associated with this clock to be end points.

  If a specified clock signal name does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and the tool makes no attempt to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, the tool considers their order left-to-right in the command string to be the order in which a signal would pass through the nodes on the specified multicycle path. For example, the following multicycle path specification:

  delete multicycle paths -from G -through H I -through J K -to L

  specifies all paths that start from G, pass through either H or I, then pass through J or K, and end at L. If you do not use the -Through argument when specifying a path, the tool will

consider all paths from the specified starting points to the specified end points to be false paths.

If a specified through node does not translate to an internal gate, the tool will transcript an error message and not delete the multicycle path definition.

Example

The following example displays current multicycle path definitions, then deletes one of them, echoing its definition as it is deleted:

report multicycle paths -all

```
// Multicycle Path -from /my_design/a -through /my_design/u1 -cycles 2
// Multicycle Path -to /my_design/c -cycles 3
// Multicycle Path -through /my_design/u24 /my_design/u25 -cycles 2
// Total reported multicycle paths = 3
```

delete multicycle paths -verbose -to /my_design/c

```
// Multicycle Path -to /my_design/c (/my_design/c/Y (130)) -cycles 2
// Deleted 1 multicycle path.
```

Refer to the add_false_paths command for examples of signal matches for pathnames specified using the asterisk (*) wildcard character.

Related Commands

add_false_paths
delete_false_paths
read_sdc
report_false_paths
report_multicycle_paths

delete_nets

Context: all contexts

Mode: insertion

Usage

delete_nets ***obj_spec*** [-silent]

Description

Removes net objects inside an instance of a design module.

You cannot delete a net that has a fanin or fanout. You can delete only dangling nets.

You cannot delete a net that is also a pin of an instance. You can delete the corresponding input pin using the delete_pins command, but the net will remain if it has objects in its fanout. You can delete the corresponding output pin using the delete_pins command, but the net will remain if it has objects in its fanin.

In general, you cannot delete a net that is driven by any RTL construct; however, an exception is that you can delete a dangling net driven by a Verilog assign statement. When you delete such a net, you also delete the assign statement.

Arguments

- ***obj_spec***

  A required value that specifies the name of a net, a Tcl list of net names or a collection of net objects as returned by the get_nets command. If the specified net object is part of a bus net, the entire bus is deleted.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to not generate error messages.

Example 1

The following example creates a connection between two pins using a specific net name. Once created, the net cannot be deleted until it is no longer used in any connection.

create_connections INSTA/I1 INSTB/O1 -net_name mynet
get_nets mynet

```
{mynet}
```

delete_net mynet

```
// Error: Cannot remove net 'mynet' in scope 'mychip' as it is being used
// in expression(s).
// Error: Error on 'delete_nets' at 'mynet'.
```

delete_connections { INSTA/I1 INSTB/O1}
delete_net mynet
get_nets mynet -silent

```
{}
```

Example 2

The following example checks for nets inside instance u1 with no fanin, disconnects their fanout and deletes them.

```
foreach_in_collection net [get_nets u1/*] {
   if {[sizeof_collection [get_fanins $net]] == 0} {
        delect_connections [get_fanouts $net -stop_on pin]
        delete_nets $net
   }
}
```

Related Commands

create_net
delete_pins
delete_nets
get_nets
get_icl_pins
get_icl_ports

delete_nofaults

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

Mode: setup

Usage

delete_nofaults {**-All** | ***modulename*** ... **-Module** [-PIN *module_pin_pathname* ...] | ***object_expression*** ... [-PIN | -Instance]} [-Stuck_at {01 | 0 | 1}] [-Class {System | User | Full}] [-VERBose] [(> | >>) *file_pathname*]

Description

Removes the nofault settings from either the specified pin or instance/module pathnames.

The delete_nofaults command deletes nofault settings you added using the add_nofaults command.

You can optionally specify nofault settings that have a specific stuck-at or transition value. For the latter, you specify 0 for “slow-to-rise” and 1 for “slow-to-fall” transition faults. If you do not specify a stuck-at (transition) value when deleting a nofault setting, the command deletes both stuck-at (transition) nofault settings.

You can use the report_nofaults command to display all the current nofault settings.

You can also optionally specify nofault settings that have a specific class code: user-defined, system netlist, or both. If you do not specify a class code, then the command deletes the nofault setting from the user class.

Arguments

- **-All**

  A switch that deletes all nofault settings.

- ***modulename***

  A repeatable string that specifies the name(s) of the module(s) from which you want to delete nofault settings. You must include the -Module switch when you specify a module name.

- **-Module**

  A switch that specifies interpretation of the *modulename* argument as a module pathname. All instances of these modules are affected. You can use the asterisk (*) and question mark (?) wildcards for the ***modulename*** argument, and the tool deletes the nofault for all matching modules or library models.

  When you follow the -Module switch with the optional -PIN switch, you can specify a list of module pin pathnames for the pin within the module. The tool automatically finds all the instances with the given module name and deletes nofaults for every pin specified in module_pin_pathname list.

- ***object_expression***

  A string representing a list of pathnames of instances or pins from which you want to delete nofault settings. You can use regular expressions, which may include any number of asterisk (*) and question mark (?) wildcard characters.

  Pin pathnames must be ATPG library cell instance pins, also referred to as design level pins. If the object expression specifies a pin within an instance of an ATPG library model, the tool ignores it. By default, pin pathnames are matched first. If a pin pathname match is not found, the tool next tries to match instance pathnames. You can force the tool to match only pin pathnames or only instance pathnames by including the -Pin or -Instance switch after the *object_expression*.

- -Pin

  An optional switch that specifies to use the preceding object expression to match only pin pathnames; the tool will then delete nofault settings from all the pins matched.

- -Instance

  An optional switch that specifies to use the preceding object expression to match only instance pathnames; the tool will then delete nofault settings from all boundary and internal pins of the instances matched.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition values from which you want to remove a nofault setting. The choices are as follows:

  01 — A literal specifying that for stuck-at faults the tool delete both “stuck-at-0” and “stuck-at-1” nofault settings; or for transition faults the tool delete both “slow-to-rise” and “slow-to-fall” nofault settings. This is the default.

  0 — A literal that deletes only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

  1 — A literal that deletes only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- -Class User | System | Full

  An optional switch and literal pair that specifies the source (or class) of the nofault settings which you want to delete. The valid literals are as follows:

  User — A literal that deletes the user-entered nofault settings. This is the default.

  System — A literal that deletes netlist-based nofault settings.

  Full — A literal that deletes all the nofault settings in the user and system classes.

- -VERbose

  By default when you specify wildcard characters for instance (-INstance) or pin (-PIN) names, the tool outputs a summary message similar to the following:

  // Note: Adding faults for 330 fault sites.

When you specify the optional -VERbose switch, then the tool outputs the instance or pin names instead of the summary. You can optionally redirect this output to a file using the > or >> redirection operators.

If you use actual instance or pin names instead of wildcards, then this switch has no effect.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example deletes nofault settings from the pins of instance i_1007 and then adds all faults to the circuit, thereby allowing the tool to add_faults to the pins of the i_1007 instance:

```
add_nofaults i_1006 i_1007 i_1008 -instance
delete_nofaults i_1007 -instance
set_system_mode analysis
add_faults -all
```

Related Commands

add_faults
add_norace_clocks
delete_faults
report_faults
report_nofaults

delete_nonscan_instances

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_nonscan_instances {{***pathname***… | ***instance_expression*** [-INStance | -Module]} | **-All**} [-Class {User | System | Full}]

Description

Removes the specified sequential instances from the non-scan instance list.

The delete_nonscan_instances command deletes sequential instances or all instances within the specified module that were previously added to the non-scan instance list by using either the add_nonscan_instances command or the Dont_touch property in a Genie netlist. You can delete either a specific list of instance names or all instances.

To display the current non-scan instance list, use the report_seq_transparent_procedures command.

Arguments

- ***pathname***

  A repeatable string that specifies either the pathnames of the instances that you want the tool to delete from the non-scan instance list.

- ***instance_expression***

  A string representing a list of instances within the design. The string instance_expression is defined as:

  { string | string * } ...

  The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. You can use a pathname expression to select several instances and the tool will then delete_nonscan_instances for all the pins on those instances; but if the expression specifies a location below the instance level of an ATPG library model, the tool will issue an error message.

- -INStance | -Module

  A switch that specifies whether the pathnames are instances or modules. An example Verilog module is “module clkgen (clk, clk_out, …)” where clkgen is the module name. The default is -Instance.

- **-All**

  A switch that specifies to delete all instances from the non-scan instance list.

- -Class User | System | Full

  An optional switch and literal pair that specifies the source (or class) of the non-scan instance that you want to delete. The valid literals are as follows:

  User — A literal that specifies to only delete the non-scan instances entered by the user using the add_nonscan_instances command. This is the default.

  System — A literal that specifies to only delete the non-scan instances described in the Genie netlist with the Dont_touch property.

  Full — A literal that specifies to delete all the non-scan instances in the user and system class.

Examples

The following example deletes an extra sequential non-scan instance called i_1007.

```
add_nonscan_instances i_1006 i_1007 i_1008
delete_nonscan_instances i_1007
insert_test_logic
```

Related Commands

add_nonscan_instances
report_seq_transparent_procedures

delete_nonscan_models

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_nonscan_models ***model_name***… | **-All** [-Class {User | System | Full}]

Description

Removes from the non-scan model list the specified sequential DFT library models.

When you issue the add_nonscan_models command on a DFT library model, the tool places all instances of that DFT library model into the user-specified, non-scan instance list. Once you remove a model from the non-scan model list with the delete_nonscan_models command, the tool then has the freedom to decide whether to place those instances of that model in the scan instance list.

The tool decides whether to place individual instances in the scan instance list based on many parameters including the scan setup settings. For example, if the scan setup has been changed to All with the command, then the tool is forced to place all available sequential instances into the scan instance list.

Arguments

- ***model_name***

  A repeatable string that specifies the model names that you want to delete from the non-scan model list. Enter the model names as they appear in the DFT library.

- **-All**

  A switch that specifies to delete all models from the non-scan model list.

- -Class User | System | Full

  An optional switch and literal pair that specifies the class code of the non-scan model that you specify. The valid literal names are as follows:

  User — A literal that specifies that the list of non-scan models were previously added by using the add_nonscan_models command. This is the default class.

  System — A literal that specifies that the list of non-scan models were added by the tool.

  Full — A literal that specifies that the list of non-scan models consist of both the user and system class.

Examples

The following example deletes an extra sequential non-scan model called d_flip_flop2:

```
add_nonscan_models d_flip_flop1 d_flip_flop2
delete_nonscan_models d_flip_flop2
insert_test_logic
```

Related Commands

add_nonscan_instances
add_nonscan_models
report_nonscan_models

delete_norace_clocks

Context: dft -edt, patterns -scan

Mode: analysis

Usage

delete_norace_clocks **-All** | ***clock_pin_name…*** | ***gate_id#…***

Description

Removes clocks from user-defined non-interacting clock domains.

The delete_norace_clocks command removes clocks from any user-defined non-interacting clock domains to which they belong. These clock domains are created originally with the add_norace_clocks command.

Arguments

- **-All**

  A switch that specifies all user-defined, non-interacting clocks previously defined with the add_norace_clocks command.

- ***clock_pin_name…***

  A repeatable string that specifies the name of a clock pin for a user-defined, non-interacting clock previously defined with the add_norace_clocks command.

- ***gate_id#…***

  A repeatable positive integer that specifies the gate identification number of a clock pin for a user-defined, non-interacting clock previously defined with the add_norace_clocks command.

Example

The following example first displays the clock domains as determined by the tool, and then displays the current user-defined non-interacting clocks and their domains. The example then removes clk4 and clk1 from their user-defined domains, and again reports the tool-defined clock domains. You can see that the user-defined clock domains do not alter the clock domains the tool determines with its own analysis.

set_system_mode analysis
report_clock_domains

```
// Clock name   Domain
// --------------------
// /clk1 (2)         1
// /clk2 (10)        2
// /clk3 (18)        3
// /clk4 (44)        4
// /clk5 (51)        5
```

report_norace_clocks

```
// User clock domain 1 : /clk1 /clk2
// User clock domain 2 : /clk3 /clk4 /clk5
```

delete_norace_clocks clk4
report_norace_clocks

```
// User clock domain 1 : /clk1 /clk2
// User clock domain 2 : /clk3 /clk5
```

delete_norace_clocks clk1
report_norace_clocks

```
// User clock domain 1 : /clk3 /clk5
```

report_clock_domains

```
// Clock name   Domain
// --------------------
// /clk1 (2)         1
// /clk2 (10)        2
// /clk3 (18)        3
// /clk4 (44)        4
// /clk5 (51)        5
```

Related Commands

add_clocks
add_norace_clocks
report_clocks
report_norace_clocks

delete_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Prerequisites: You must add circuit points with the add_notest_points command before you can delete them.

Usage

delete_notest_points {***pin_pathname***… | ***instance_pathname…*** | ***instance_expression*** } | **-ALL** | {**-Path** ***critical_pathname***} | **-ALL_Paths**

Description

Removes the circuit point which the tool cannot use for testability insertion from the specified pins.

The delete_notest_points command deletes circuit points added using the add_notest_points command. These notest circuit points identify output pins of cells that the tool is not to use for insertion of controllability and observability. You can display a list of the current circuit points and their associated pins by using the report_notest_points command.

Arguments

- ***pin_pathname***

  A repeatable string that specifies a list of pins for which you want to delete the circuit points that the tool cannot use for testability insertion.

- **-All**

  A switch that deletes all previously-added circuit points.

- ***instance_pathname***

  A required repeatable string that lists the instances whose output pins you want to delete the circuit points that the tool cannot use for testability insertion. All output pins within that (hierarchical) instance are removed from the list of pins that should be excluded from consideration.

- ***instance_expression***

  A string representing a list of instances within the design. The string *instance_expression* is defined as:

  { string | string * } ...

  The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. You can use a pathname expression to select several instances and the tool will then delete_notest_points for all the pins on those instances; but if the expression

specifies a location below the instance level of an ATPG library model, the tool will issue an error message.

- **-Path *critical_pathname***

  A required switch and name pair that specifies to delete the named critical path. You can list the names of the critical paths using the report_notest_points command with the -Paths switch. For more information on the format of the file, refer to "The Path Definition File" in the *Tessent Scan and ATPG User's Manual*.

- **-ALL_Paths**

  A required switch that specifies to delete all critical paths.

Example

The following example deletes an incorrect notest circuit point and corrects it with a new circuit point before performing testability analysis:

```
set_system_mode analysis
add_notest_points tr_i ts_i
delete_notest_points tr_i
add_notest_points tr_io
```

Related Commands

add_output_masks

report_notest_points

delete_output_masks

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

Mode: setup

Usage

delete_output_masks ***primary_output***... | **-All**

Description

Removes the masking of the specified primary output pins.

The tools use primary output pins as the observe points during the fault detection process. When you mask a primary output pin with the add_output_masks command, the tools mark that pin as an invalid primary output during the fault detection process.

You can set a default mask for all output and bidirectional pins using the set_output_masks command. You can add a hold value to a default mask with the add_output_masks command, or remove a hold value using the delete_output_masks command. To turn off the default masks for all output pins, you must use the "set_output_masks Off" command.

Arguments

- ***primary_output***

  A repeatable string that specifies the names of the primary output pins that you want to unmask.

- **-All**

  A switch that unmasks all primary outputs masked using the add_output_masks command.

Example

The following example first incorrectly chooses two of the design's primary output pins to mask. The example then unmasks the one primary output that was inappropriate, masks the correct primary output, and then displays the complete list of currently-masked primary output pins no longer available as observation points:

```
add_output_masks q1 qb3
delete_output_masks q1
add_output_masks qb1
report_output_masks
```

```
qb1
qb3
```

Related Commands

add_output_masks
report_output_masks

delete_patterns

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

Mode: analysis

Usage

For patterns -scan

delete_patterns [-external | -internal] [-pattern_sets [chain] [scan]] [-keep_fault_status] [-force]

For patterns -ijtag

delete_patterns [-pattern_sets *pattern_set_list*] [-force]

For patterns -scan_retargeting

delete_patterns **core_name *name*** [-mode *name*] [-silent]

Description

Deletes the current pattern set.

This command removes either the internal or external pattern set or both from the current session.

You can only delete generated pattern sets (that is, patterns generated by the tool and not read in) that have been written out using the write_patterns command, unless you use the -force switch. This restriction prevents you from deleting patterns that you have generated but have not yet written out to disk. In the -ijtag context, you can delete only patterns that have been closed.

Arguments

- -external | -internal **(patterns -scan context only)**

  Optional switches that specify which pattern source to remove. By default, the command removes both internal and external pattern sets.

- -pattern_sets [chain] [scan] **(patterns -scan only)**

  An optional switch and keyword pair that specifies which pattern type (scan or chain) to remove from the current session. By default, the tool removes both scan and chain patterns.

- -pattern_sets *pattern_set_list* **(-ijtag and patterns -scan_retargeting context only)**

  An optional switch and string that specifies a Tcl list of pattern set names that were created using the open_pattern_set command. When you don't use this switch, the command deletes all pattern sets.

- -keep_fault_status **(-scan context only)**

  Optional switch that keeps the current fault status when removing the internal patterns. When deleting the internal patterns, the status of existing faults is reset by default if you don't use this switch.

- -force

  An optional switch that forces removing the internal patterns even when they are not stored previously. By default, the tool does not allow you to remove the unsaved internal patterns. However, the -force option allows you to delete the unsaved internal patterns without this safeguard. You can always remove the external patterns.

- core_name name **(patterns -scan_retargeting only)**

  A required switch that specifies the cores from which pattern sets are to be deleted. If the core *name* does not exist, the patterns for all cores are deleted in the pattern retargetable phase.

- -mode name **(patterns -scan_retargeting only)**

  An optional switch that specifies which modes of the core the patterns should be deleted from. If -mode is not specified, all modes of the core are deleted.

- -silent **(patterns -scan_retargeting only)**

  An optional switch that specifies not to report patterns as they are deleted. By default, deleted patterns are reported.

Example 1

The following example removes all pattern sets currently in memory:

```
delete_patterns
```

Example 2 (-scan context)

The following example removes only the chain patterns from the internal pattern set. The -force option suppresses the error that would otherwise occur if you had not already saved the internal patterns using the write_patterns command.

```
delete_patterns -internal -pattern_sets chain -force
```

Example 3 (-ijtag context)

The following example removes only the patterns in the Tcl list named "pattern_list5":

```
delete_patterns -pattern_sets patterns_list5
```

Related Commands

read_patterns
simulate_patterns
write_patterns
open_pattern_set

delete_pins

Context: all contexts

Mode: insertion

Usage

delete_pins ***obj_spec*** -silent

Description

Removes pin objects from a design module instance.

If the pin to be deleted has an internal connection, the internal nets associated to the pin are kept; otherwise they are removed too.

Note

You should be aware that deleting one bit of a bus deletes the entire bus.

Arguments

- ***obj_spec***

  A required value that specifies a pin name, a Tcl list of pin names, or a collection of pin objects as returned by the get_pins command. If the specified pin object is part of a bus net, the entire bus is deleted.

- -silent

  An optional argument that specifies to not generate an error message if an object within the specified ***obj_spec*** does not exist.

Example 1

The following example deletes the specified pins names. On instance u1, pin A is a scalar port, pin B is a bus pin with indices [3:0], and port C is also a bus port with indices [7:0]. All three pins are completely deleted even though the complete bus range was not specified for pins B and C.

get_pins {u1/A u1/B u1/C}

```
{u1/A u1/B[3] u1/B[2] u1/B[1] u1/B[0] u1/C[7] u1/C[6] u1/C[5]
u1/C[4]u1/C[3] u1/C[2] u1/C[1] u1/C[0]}
```

delete_pins {u1/A u1/B u1/C[1]}
get_pins {u1/A u1/B u1/C}

```
//  Warning: No result. Please check your filtering options.
{}
```

Example 2

The following example deletes the collection of pin objects returned by the get_pins command.

delete_pins [get_pins {u1/A u1/B u1/C[1]} -silent]

Related Commands

create_pin
get_icl_pins
get_icl_ports
get_nets

delete_ports

Context: all contexts

Mode: insertion

Usage

delete_ports ***obj_spec1*** [-on_module *obj_spec2*] [-silent]

Description

Removes port objects from a design module.

If the port to be deleted has an internal connection, the internal nets associated to the port are kept; otherwise they are removed too.

If obj_spec1 is a collection of ports, those names refers to ports on specific modules and the specified -on_module option is ignored.

Note

Deleting one bit of a bus port deletes the entire bus port.

Arguments

- ***obj_spec1***

  A required value that specifies a single port name, a Tcl list of port names or a collection of port objects.

- -on_module *obj_spec2*

  An optional switch and value pair that specifies the module from which the ports are to be deleted. The obj_spec2 value can be the name of a module or a module object returned by the command. If the -on_module switch is not specified, this command deletes the port from the current design which is the module specified by the set_current_design command. You cannot delete a port on a module that was read in from a file and that is not instantiated inside the current design. It is possible to delete ports on modules created with the create_module command.

- -silent

  An optional argument that specifies to not generate an error message if an object within the specified obj_spec1 or obj_spec2 does not exist.

Example 1

The following example deletes the specified port names of module ModA. On ModA, port A is a scalar port, port B is a bus port with indices [3:0], and port C is also a bus port with indices [7:0]. All three ports are completely deleted even if the complete bus range was not specified for port B and C.

```
delete_ports {A B C[1]} -on_module ModA
```

Example 2

The following example deletes the collection of port objects returned by the get_ports command. Notice that the -on_module option is not specified for the delete_ports command because the port objects returned by the get_ports command are already associated to a specific module.

```
delete_ports [get_ports {A B C[1]} -of_module ModA]
```

Related Commands

create_port

get_icl_ports

delete_primary_inputs

Context: all contexts

Mode: setup

Usage

delete_primary_inputs {***net_pathname***... | ***primary_input_pin***... | **-All**}
[-Class {User | System | Full}]

Description

Removes the specified primary inputs from the existing flattened model or, if a flattened model does not yet exist, from the flattened result.

The delete_primary_inputs command deletes from a circuit the primary inputs that you specify. You can delete either the user class, system class, or full classes of primary inputs. If you do not specify a class, the tool deletes the primary inputs from the user class.

You can display a list of any class of primary inputs by using the report_primary_inputs command.

Note

Once you delete a primary input pin with this command, adding the pin back again with the add_primary_inputs command will not create the same pin. Bus information is lost and the pin will be marked as user-added. Take care to use delete_primary_inputs only if you wish to truly delete a pin from the design so it is not used.

Arguments

- ***net_pathname***

  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- ***primary_input_pin***...

  A repeatable string that specifies a list of primary input pins that you want to delete. You can specify the class of primary inputs to delete with the -Class switch.

- **-All**

  A switch that deletes all primary inputs. You can specify the class of primary inputs to delete with the -Class switch.

- -Class User | System | Full

  An optional switch and literal pair that specifies the class code of the designated primary input pins. The valid class code literal names are as follows:

  User — A literal specifying that you added the primary inputs using the add_primary_inputs command. This is the default class.

  System — A literal specifying that the primary inputs derive from the netlist.

Full — A literal specifying that the primary inputs consist of both user and system classes.

Example

The following example deletes an extra added primary input from the user class of primary inputs:

```
add_primary_inputs indata2 indata4 indata6
delete_primary_inputs indata4 -class user
```

Related Commands

add_primary_inputs

report_primary_outputs

delete_primary_outputs

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

Mode: setup

Usage

delete_primary_outputs {***net_pathname***... | ***primary_output_pin***... | **-All**} [-Class {User | System | Full}]

Description

Removes the specified primary outputs from the existing flattened model or, if a flattened model does not yet exist, from the flattened result.

The delete_primary_outputs command deletes from a circuit the primary outputs that you specify. You can delete either the user class, system class, or full classes of primary outputs. If you do not specify a class, the tool deletes the primary outputs from the user class.

You can display a list of any class of primary outputs by using the report_primary_outputs command.

Note

Once you delete a primary output pin with this command, adding the pin back again with the add_primary_outputs command will not create the same pin. Bus information is lost and the pin will be marked as user-added. Take care to use delete_primary_outputs only if you wish to truly delete a pin from the design so it is not used.

Arguments

- ***net_pathname***

  A repeatable string that specifies the circuit connections that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- ***primary_output_pin***

  A repeatable string that specifies a list of primary output pins that you want to delete. You can specify the class of primary outputs to delete with the -Class switch.

- **-All**

  A switch that deletes all primary outputs. You can specify the class of primary outputs to delete with the -Class switch.

- -Class User | System | Full

  An optional switch and literal pair that specifies the class code of the primary output pins that you specify. The valid literal names are as follows:

  User — A literal specifying that the list of primary outputs were added using the add_primary_outputs command. This is the default class.

System — A literal specifying that the list of primary outputs derive from the netlist.

Full — A literal specifying that the list of primary outputs consists of both the user and system class.

Example

The following example deletes a primary output from the system class of primary outputs:

```
delete_primary_outputs outdata1 -class system
```

Related Commands

add_primary_outputs

report_primary_outputs

delete_processors

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion (dft -edt context only)

Usage

delete_processors [*hostname*[:*cpus*]]…

Description

Removes processor definitions previously specified with the add_processors command.

The delete_processors command deletes slave processor definitions you previously specified using the add_processors command. When you issue the command with no arguments, the tool discontinues all current slave processes. If you specify one or more *hostname* arguments, the tool will discontinue slave processes only on the specified host machines. If you include :*cpus* with *hostname*, the tool will discontinue only the specified number of slave processes on that host machine. Without the :*cpus*, all current slave processes on that host machine are discontinued. When specifying *hostname*, you must specify the network name of the machine, even if it was originally selected by an LSF, SGE, or custom job scheduler.

Be aware that the process run by an LSF, SGE, or custom job scheduler is a script rather than the actual slave process. If you discontinue these slave processes with the delete_processors command, the corresponding scripts also terminate.

Tip: Alternatively, you can end execution of the scripts by using the job scheduler commands (SGE's **qdel**, LSF's **bkill**, or whatever is used by the job scheduler invoked by the add_processors Generic command) or a Linux **kill** command. There will be a small delay before the slave process is actually shut down to allow for the detection of the script death and for the shutdown to be coordinated with the master process. Slave processes shut down in this manner will appear to have died in the master process's transcript. This is normal behavior. Detection of script deaths occurs only during the add_processors, delete_processors, and report_processors commands. You can force detection from the command line without other side effects by issuing the report_processors command with no arguments.

Arguments

- *hostname*

  A repeatable string that specifies a host machine you previously defined with the add_processors command or received via a job scheduler. The hostname choices are as follows:

  machine_name — A string that specifies the host machine by its network name.

  LOCALHOST — A literal that specifies the host machine is the machine on which you invoked the tool.

  IPaddress — A string that specifies the host machine by its IP address.

- : *cpus*

  An optional colon (:) and positive integer pair that specifies the number of processes to delete on the particular host machine.

Example

The following example begins by reporting the distributed ATPG status. It then discontinues all the slave processes currently running on one remote host (nemo) and one of the slave processes running on the other remote host (ahab):

report_processors

```
hosts/processors   arch      CPU(s)     %idle    free RAM    process size
----------------  -------  -----------  -----  ------------  ------------
ace   (master)    x86      2 x 2.2 GHz    81%    8393.52 MB      20.07 MB
nemo           1  x86      2 x 2.0 GHz    98%     382.33 MB      18.78 MB
               2                                               18.78 MB
               3                                               18.78 MB
               4                                               18.78 MB
ahab           1  x86      1 x 1.7 GHz    99%     334.98 MB      18.78 MB
               2                                               18.78 MB
master and 6 processors running.
```

delete_processors nemo ahab:1

```
//  Deleted host nemo:4
//  Deleted host ahab:1
//  Total processors: 1
//  Note: Freeing 1 'Tessent FastScan' license.
```

report_processors

```
hosts/processors   arch      CPU(s)     %idle    free RAM    process size
----------------  -------  -----------  -----  ------------  ------------
ace   (master)    x86      2 x 2.2 GHz    81%    8393.52 MB      20.07 MB
ahab           1  x86      1 x 1.7 GHz    97%     229.83 MB      18.78 MB
master and 1 processor running.
```

Related Commands

add_processors
compress_patterns
create_patterns
identify_redundant_faults
order_patterns
report_processors

delete_read_controls

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

Mode: setup

Usage

delete_read_controls ***primary_input_pin***... | **-All**

Description

Removes the read control line definitions from the specified primary input pins.

The delete_read_controls command deletes read control lines defined with the add_read_controls command. You can delete the read control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***

  A repeatable string that specifies a list of primary input pins from which you want to delete any read control line definitions.

- **-All**

  A switch that deletes the read control line definitions for all primary input pins.

Example

The following example deletes an incorrect read control line, then redefines that read control line with the correct off-state:

```
add_read_controls 0 r1 r2
delete_read_controls r1
add_read_controls 1 r1
set_system_mode analysis
```

Related Commands

add_read_controls

report_read_controls

delete_register_value

Context: all contexts

Mode: setup

Usage

delete_register_value { **-All** | ***value_name*** ...}

Description

Deletes all or a specified register value variable specified with the add_register_value command.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- **-All**

  A required switch that deletes all register value variables.

- ***value_name*** ...

  A required repeatable string that deletes a named ***value_name***.

Example

The following example deletes all register value variables:

```
delete_register_value -all
```

Related Commands

add_register_value
set_register_value
report_register_value

delete_scan_chains

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

Mode: setup

Usage

delete_scan_chains ***chain_name***... | **-All**

Description

Removes the specified scan chain definitions from the scan chain list.

The delete_scan_chains command deletes scan chains defined with the add_scan_chains command. You can delete the definitions of specific scan chains or of all scan chains.

dft -scan Details

When you remove a scan chain definition, it is only the definition you are removing, not the scan chain itself.

Arguments

- ***chain_name***

  A repeatable string that specifies the names of the scan chain definitions that you want to delete.

- **-All**

  A switch that deletes all scan chain definitions.

Example

The following example defines several scan chains, adding them to the scan chain list, then deletes one of the scan chains:

```
add_scan_chains chain1 group1 indata2 outdata4
add_scan_chains chain2 group1 indata3 outdata5
add_scan_chains chain3 group1 indata4 outdata6
delete_scan_chains chain2
```

Related Commands

add_scan_chains

report_scan_chains

delete_scan_groups

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

Mode: setup

Usage

delete_scan_groups ***group_name***... | **-All**

Description

Removes the specified scan chain group definitions from the scan chain group list.

The delete_scan_groups command deletes scan chain groups defined with the add_scan_groups command. You can delete the definitions of specific scan chain groups or of all scan chain groups.

When you delete a scan chain group, the tool also deletes all scan chains within the group.

Arguments

- ***group_name***

  A repeatable string that specifies the names of the scan chain group definitions that you want to delete.

- **-All**

  A switch that deletes all the scan chain group definitions.

Example

The following example defines two scan chain groups, adding them to the scan chain group list, then deletes one of the scan chain groups:

```
add_scan_groups group1 scanfile1
add_scan_groups group2 scanfile2
delete_scan_groups group1
```

Related Commands

add_scan_groups

report_scan_groups

delete_scan_instances

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_scan_instances {***pathname***… | ***instance_expression*** [-INStance | -Control_signal | -Module]} | **-All**

Description

Removes the specified, sequential instances from the user-identified scan instance list.

The delete_scan_instances command deletes sequential instances, instances specified by control signals, or all instances within the specified module that were previously added to the scan instance list by using the add_scan_instances command. You can delete either a specific list of instance names or all instances.

You can also use this command to remove an exception status from an instance, that you added using the add_scan_instances command, which allowed the tool to convert the instance into a scan cell even though it violated the S4 rule. In this case, although the instance is converted back into a non-scan instance, the effects of its previous unknown state on downstream instances is not reset.

User-identified scan instances result from using the add_scan_instances or add_scan_models commands.

If you issue an insert_test_logic command after removing an instance from the user-identified scan list with the delete_scan_instances command, the tool then has the option of including it in the system-identified scan instance list.

Arguments

- ***pathname***

  A repeatable string that specifies the pathnames of the instances or control signals (that control instances) that you want the tool to delete from the user-identified scan instance list. The pathnames must be user-identified scan instances or control signals (that control instances), which you previously selected with the add_scan_instances command.

- ***instance_expression***

  A string representing a list of instances within the design. The string instance_expression is defined as:

  { string | string * } ...

  The asterisk (*) is a wildcard that allows you to match many instances in a design. Any expression that does not contain an asterisk (*) will match exactly zero or one instance.

  This argument does not support pathnames to objects below the instance level of an ATPG library model. You can use a pathname expression to select several instances and the tool will then delete_scan_instances for all the pins on those instances; but if the expression

specifies a location below the instance level of an ATPG library model, the tool will issue an error message.

- -INStance | -Control_signal | -Module

  A switch that specifies whether the pathnames are instances, pins (control signals), or modules. An example Verilog module is “module clkgen (clk, clk_out, …)” where clkgen is the module name. You can only use the -*Control_signal* option in analysis mode. The default is -Instance.

- **-All**

  A switch that specifies to delete all instances from the user-identified scan instance list. This switch does not affect the instances in the system-identified scan instance list.

Examples

The following example deletes an extra sequential scan instance that was defined to be treated as a scan cell; thus, the deleted instance is no longer included in the user-identified scan instance list:

```
add_scan_instances i_1006 i_1007 i_1008
delete_scan_instances i_1007
```

Related Commands

add_scan_instances

report_seq_transparent_procedures

delete_scan_models

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_scan_models ***model_name***… | **-All**

Description

Removes the specified sequential models from the scan model list.

The delete_scan_models command deletes all instances of the sequential models that you specify. This includes removing all instances of the model_name from the user-identified scan instance list. You can delete a specific list of sequential models or all the models.

There are two ways that sequential instances can be identified for scan: system and user. You can explicitly identify scan instances with either the add_scan_instances or the add_scan_models commands (user-identified). The tool also selects the sequential instances automatically by targeting as many candidates as possible

If you issue an insert_test_logic command after removing a model from the user-identified scan list with the delete_scan_models command, the tool then has the option of including any of the instances of that model in the system-identified scan instance list.

To display the current scan model list, use the report_scan_models command.

Arguments

- ***model_name***

  A repeatable string that specifies the model names that you want to delete from the scan model list and user-identified scan instance list. Enter the model names as they appear in the DFT library.

- **-All**

  A switch that specifies to delete all models from the scan model list and all instances from the user-identified scan instance list.

Examples

The following example deletes an extra added sequential scan model; thus, the deleted model is no longer included in the scan model list:

```
add_scan_models d_flip_flop1 d_flip_flop2
delete_scan_models d_flip_flop2
```

Related Commands

add_scan_instances
add_scan_models
report_scan_models

delete_scan_partitions

Context: dft -scan

Mode: setup, analysis

Usage

delete_scan_partitions {***object_name...*** | **-All**}

Description

Deletes the user-specified scan partitions.

It does not delete the default scan partition, default_scan_partition. The sequential cells whose scan partitions are deleted are placed back into the default scan partition.

Arguments

- ***object_name***

  A required repeatable string that specifies the names of the scan partitions to remove.

- **-All**

  A required switch that specifies to remove all the user specified scan partitions. This option does not remove the default scan partition that the tool creates, default_scan_partition.

Examples

The following example adds three scan partitions. The scan partitions are then deleted, first partially, then completely.

```
add_scan_partition  part_1  -instance umodA
add_scan_partition  part_2  -instance umodB umodC
add_scan_partition  part_3  -instance umodD
delete_scan_partitions  part_2
report_scan_partitions

----------------------------------------------------------------------
ScanPartitionName    TotalNumCells/ScannableCells  Members
----------------------------------------------------------------------
part_1                         150/150           umodA [instance]
part_3                         330/250           umodD [instance]
default_scan_partition       15000/0             <all_remaining_cells>
----------------------------------------------------------------------

delete_scan_partitions  -all
report_scan_partitions

----------------------------------------------------------------------
ScanPartitionName    TotalNumCells/ScannableCells  Members
----------------------------------------------------------------------
default_scan_partition        15480/400           <all_remaining_cells>
----------------------------------------------------------------------
```

Related Commands

add_scan_partition

report_scan_partitions

delete_scan_pins

Context: dft -scan

Mode: setup, analysis

Usage

delete_scan_pins ***chain_name***… | **-All**

Description

Removes any previously assigned scan input, output, and clock names from the specified scan chains.

The delete_scan_pins command removes user-specified names of the scan input and scan output pins of the scan chains that you previously assigned with the add_scan_pins command. You can use the report_scan_pins command to display all added scan input and output pin names for each scan chain.

Arguments

- ***chain_name***

  A repeatable string that specifies the names of the scan chains from which you want the tool to remove the associated pins.

- **-All**

  A switch that removes all added scan pin names from all scan chains.

Examples

The following example removes previously-assigned scan chain input and output names for chain1:

```
add_clocks 0 clk
set_system_mode analysis
add_scan_pins chain1 si so
add_scan_pins chain2 si1 so1
delete_scan_pins chain1
insert_test_logic
```

Related Commands

report_scan_pins

delete_simdut_fault

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

Mode: setup, analysis

Usage

delete_simdut_fault **-signal** ***signal_name***

Description

Removes the injected stuck-at fault at the specified signal.

Arguments

- **-signal** ***signal_name***

  A required switch and string pair that specifies the name of a signal from the Verilog netlist.

Example

The following example removes an injected stuck-at fault from the SIMDUT_TB.CHIP.memory00.q_a signal.

```
delete_simdut_fault -signal SIMDUT_TB.CHIP.memory00.q_a
```

Related Topics

add_simdut_fault

delete_simulation_contexts

Context: all contexts

Mode: setup, analysis

Prerequisite: A flat model must currently exist

Usage

delete_simulation_contexts {*context_name*... | -all_user_defined }

Description

Deletes one or more user-defined simulation contexts.

The delete_simulation_contexts command allows you to delete one or multiple or all user-defined simulation contexts.

Arguments

- *context_name*

  An optional string that defines the name of one or more simulation contexts. Note that you cannot use wildcard characters in the context_name. You also cannot delete any of the predefined context names listed in Table 7-40 on page 1648.

- -all_user_defined

  An optional switch that specifies all user-defined simulation contexts.

Example 1

The following example lists all of the user-defined simulation contexts and then deletes them:

```
SETUP> get_simulation_context_list  -user_defined
{myContext1 myContext2 myContext3}

SETUP> delete_simulation_contexts -all_user_defined
SETUP> get_simulation_context_list -user_defined
{ }
```

Example 2

The following example lists all of the user-defined simulation contexts and then deletes two of them:

```
SETUP> get_simulation_context_list  -user_defined
{myContext1 myContext2 myContext3}

SETUP> delete_simulation_contexts { myContext2 myContext3 }
SETUP> get_simulation_context_list -user_defined
{ myContext1 }
```

Related Commands

add_simulation_context
copy_simulation_context
get_current_simulation_context
get_simulation_context_list
report_simulation_contexts
set_current_simulation_context

delete_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

delete_simulation_forces {***obj_spec*** **|** -all }

Description

Removes forces from one or more gate_pin objects.

The delete_simulation_forces command removes simulation forces from the specified gate_pin object(s). When this command removes a simulation force, the tool immediately propagates the values on the driving gate_pin objects to keep the circuit in a consistent state.

This command is allowed for user-defined contexts only.

Arguments

- ***obj_spec***

  An optional value that specifies one or more gate_pin objects.

- -all

  An optional switch that removes all simulation forces on all gate_pin objects within the current simulation context.

Example

The following example shows the effect of adding a force and then removing all forces:

ANALYSIS> **set_current_simulation_context myContext**
ANALYSIS> **add_simulation_force [get_gate_pins I1] -value 1**
ANALYSIS> **report_simulation_forces**

```
Value   Object
------  ------
1       I1
```

ANALYSIS> **delete_simulation_forces -all**
ANALYSIS> **report_simulation_forces**

Related Commands

add_simulation_force
get_simulation_value_list
report_simulation_forces
simulate_clock_pulses
simulate_forces

delete_sub_chains

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_sub_chains **-All** | {***object_name*** [*subchain_name*]
[-Module | -Instance | -Library_model]}

Description

Removes scan subchains previously defined with the add_sub_chains command. Use the report_sub_chains command to display the currently defined scan subchains.

Arguments

- **-All**

  A switch that removes all scan subchains.

- ***object_name***

  A string that specifies either the pathname of an instance, or the name of a module or library model. If you specify an instance, you must also use the -Instance switch. If you specify a library model, you must also use the -Library_model switch.

- *subchain_name*

  An optional string that specifies the name of a pre-existing scan subchain to delete. If you do not specify a *subchain_name*, all subchains in the specified *object_name* are deleted.

- -Module

  An optional switch that specifies the object_name is a module. This is the default.

- -Instance

  An optional switch that specifies the object_name is an instance.

- -Library_model

  An optional switch that specifies the object_name is a library model.

Examples

The following example deletes the subc1 subchain from the blkA module:

```
add_sub_chains blkA subc1 si1 so1 10 mux_scan -sen_core sen1 -subclock sclk -module
add_sub_chains blkA subc2 si2 so2 30 mux_scan -sen_core sen2 -subclock mclk -module
delete_sub_chains blkA subc1
```

Related Commands

add_sub_chains

report_sub_chains

delete_subchain_clocks

Context: dft -scan, dft -test_points

Mode: setup

Usage

delete_subchain_clocks ***subchain_name clock_name***

Description

Deletes subchain clocks previously defined with the add_subchain_clocks command.

Arguments

- ***subchain_name***

  A required string that specifies the name of a subchain.

- ***clock_name***

  A required string that specifies the name of a subchain clock.

Examples

The following example adds, reports, and deletes a subchain clock:

add_sub_chains MULTIBITSFF3 chain1 SI SO 4 mux_scan S library_model
add_subchain_clocks chain1 0 RESET -reset
add_subchain_clocks chain1 0 SET -set
add_subchain_clocks 0 CK -first_cell_clock -leading_edge
add_subchain_clocks 0 CK -last_cell_clock -leading_edge
report_subchain_clocks chain1

```
// clock name type         off_state   edge
// ---------- ----         ---------   -----
// RESET     reset             0
// SET       set               0
// CK        first cell clock  0       leading edge
// CK        last cell clock   0       leading edge
```

delete_subchain_clocks chain1 CK
report_subchain_clocks chain1

```
// clock name type  off_state  edge
// ---------- ----- --------- ------
// RESET       reset 0
// SET         set   0
```

Related Commands

add_subchain_clocks

report_subchain_clocks

delete_subchain_groups

Context: dft -scan

Mode: setup, analysis

Usage

delete_subchain_groups ***subchain_group_name*** … | **-All**

Description

Removes a scan subchain group.

The delete_subchain_groups command removes subchain groups added by means of the add_subchain_group command. To determine whether there are any current scan subchain groups, use the report_subchain_groups command.

Arguments

- ***subchain_group_name***

  A repeatable string that specifies the names of the subchain group definitions to be deleted.

- **-All**

  A switch that removes all subchain group definitions.

Examples

The following example removes a subchain group definition:

add_subchain_group subchaingroup1 subchain1 subchain2 -fixed
add_subchain_group subchaingroup2 subchain3 subchain4 -flexible
report_subchain_groups

```
-----------------------------------------------
SubchainGroupName   Type      ListOfSubchains
-----------------------------------------------
subchaingroup1      fixed     subchain1
                              subchain2
subchaingroup2      flexible  subchain3
                              subchain4
```

delete_subchain_group subchaingroup1

Related Commands

add_subchain_group
add_sub_chains
delete_sub_chains
report_sub_chains
report_subchain_groups

delete_test_points

Context: dft -test_points

Mode: setup

Usage

delete_test_points [**-control** | **-observe**] [**-location** ***pin/port_spec*** | **-all**]

Description

Removes the specified test point definitions.

If you do not specify -control or -observe with either the -location *pin/port_spec* or the -all option, the tool removes all the control and observe test points. If you enter the command without any arguments, the tool removes all the control and observe test points.

This command removes both user-defined and system-defined test points. You can create user-defined test points with the add_control_points and add_observe_points commands.

Arguments

- **-all**

  An optional switch that removes the control and/or observe test points from all locations.

- **-control**

  An optional switch that removes the control test points from the specified locations.

- **-observe**

  An optional switch that removes the observe test points from the specified locations.

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

  An optional switch and string that specifies a pathname to the control and/or observe test points to delete.

  pin/port spec can be the name of a pin/port, a Tcl list of pins/ports, or a collection of pin/port objects.

Examples

The following example removes test points:

```
delete_test_points /I_6_16/cp /I_7_16/q
```

The delete_test_points command only specifies the pathnames of the test points, not the type. This example includes both control and observe test points and deletes them by default.

Related Commands

add_tied_signals
report_test_logic
report_test_points
add_input_constraints
set_test_point_insertion_options

delete_tied_signals

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

Mode: setup

Usage

delete_tied_signals {***floating_object_name***... | **-All**} [-Pin]

Description

Removes the assigned (tied) value from the specified floating nets or pins.

The delete_tied_signals command deletes the tied values assigned with the add_tied_signals command. You can display a list of tied signals using the report_tied_signals command.

Whenever you delete tied values from nets or pins, be sure to re-add any necessary values before performing another simulation. If you do not add required tied values to floating nets or pins, the tool displays a warning. The warning states that the design has floating nets or pins and assumes they are tied to the default value; you must set the default value using the set_tied_signals command.

Arguments

- ***floating_object_name***

  A repeatable string that specifies the names of the tied floating nets or pins whose tied values you want to delete.

  If you do not specify the -Pin option, the tool assumes floating_object_name is a net name. If you specify a full net pathname, the tool deletes only the specified instance-based blackbox tied signal. If you specify the -Pin option, it assumes the floating_object_name is a pin name.

- **-All**

  A switch that deletes the tied values from all tied floating nets or pins. This also includes all instance-based blackbox tied signals.

- -Pin

  A switch specifying that the *floating_object_name* argument that you provide is a floating pin name.

Example

The following example deletes the tied value from the user-class tied net “vcc”; thereby leaving “vcc” as a floating net:

```
add_tied_signals 1 vcc vdd
delete_tied_signals vcc
```

Related Commands

add_tied_signals
delete_black_boxes
report_tied_signals
set_tied_signals

delete_write_controls

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

Mode: setup

Usage

delete_write_controls ***primary_input_pin***... | **-All**

Description

Removes the write control line definitions from the specified primary input pins.

The delete_write_controls command deletes write control lines defined with the add_write_controls command. You can delete the write control line definitions for specific pins or for all pins.

Arguments

- ***primary_input_pin***

  A repeatable string that specifies a list of primary input pins from which you want to delete any write control line definitions.

- **-All**

  A switch that deletes the write control line definitions for all primary input pins.

Example

The following example deletes an incorrect write control line, then re-adds that write control line with the correct off-state:

```
add_write_controls 0 w1 w2
delete_write_controls w1
add_write_controls 1 w1
set_system_mode analysis
```

Related Commands

add_write_controls

report_write_controls

diagnose_failures

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: The read_patterns command must be set to a pattern source.

Usage

diagnose_failures {***failure_filename*** | **-Internal**} [-Output *report_filepath*] [-Csv *csv_filename*] [-Replace] [-Faulty_chain {*faulty_chain_name faulty_chain_type*}...]

Description

Diagnoses the failures in the specified failure file.

The diagnose_failures command performs a diagnosis of the failing scan and chain test patterns in the specified failure file, or diagnosis of the IDDQ patterns in the specified failure file.

For scan and chain test patterns, you must translate the ATE failure log into a failure file as described in the "Guidelines for Preparing the ATE Failure File" section in the *Tessent Diagnosis User's Manual*. For IDDQ patterns, see "IDDQ Failure File Format" for the required failure file format.

It is critical that the data in the test patterns, flattened netlist, and failure file be synchronized for diagnosis. For more information, see "Input File Requirements" in the *Tessent Diagnosis User's Manual*.

For complete information on running diagnosis and interpreting the results, see the *Tessent Diagnosis User's Manual*.

Arguments

- ***failure_filename***

  A required string that specifies the name of the file that contains the failing test pattern information for diagnosis. This file can use either the pattern-based or cycle-based format as described in the "Guidelines for Preparing the ATE Failure File" section in the *Tessent Diagnosis User's Manual*.

  For IDDQ diagnosis, specify the IDDQ failure file. See "IDDQ Diagnosis" for more information.

- **-Internal**

  An required switch that directs the tool to use the in-memory failure file. You must issue a read_failures command before you use the diagnose_failures -internal command. See "Diagnosis Improvements Through Additional Pattern Generation" in the *Tessent Diagnosis User's Manual* for complete information.

- -Output *report_filepath*

  An optional switch and string pair that directs the tool to write the diagnostic report in ASCII format to the specified file. By default, the diagnostic report displays on screen.

- -Csv *filename*

  An optional switch and string pair that directs the tool to write the diagnostic report in Comma Separated Value (CSV) format to the specified file. The CSV format does not include the list of failing patterns for each symptom. By default, the diagnostic report displays on screen.

- -Replace

  An optional switch that directs the tool to overwrite existing diagnostic report files in either ASCII or CSV format. By default, the tool will not overwrite existing files.

- -Faulty_chain *faulty_chain_name faulty_chain_type*

  An optional switch and repeatable string and literal pair that allows you to run diagnosis on specified chain failures in addition to the scan failures contained in a failure file.

  faulty_chain_name — string that specifies the name of the faulty chain.

  fault_type — literal that specifies the fault type associated with the faulty chain. Valid fault types include:

  STUCK_AT_1

  STUCK_AT_0

  SLOW_TO_FALL

  SLOW_TO_RISE

  SLOW

  FAST_TO_RISE

  FAST_TO_FALL

  FAST

  INDETERMINATE — specifies the fault type can not be determined.

Example 1

The following example: sets *pattern_file1* as the external test pattern source, injects a stuck-at-0 fault into the design, writes out a failure file named *fail_patterns*, and performs a diagnosis of the failing patterns in the *fail_patterns* file.

```
set_system_mode analysis
read_patterns pattern_file1
write_failures fail_patterns i_1006/i1 -stuck_at 0

//  failing_patterns=9  simulated_patterns=36  simulation_time=0.00 sec

diagnose_failures fail_patterns

...
```

Still using the external test pattern source file *pattern_file1*, the following example: injects an additional stuck-at-0 fault into the *fail_patterns* failure file, performs a diagnosis of the failing patterns in the *fail_patterns* file, and sends the report to a file named *fail_diags*.

write_failures fail_patterns i_1005/i1 -stuck_at 0 -replace

```
//  failing_patterns=4  simulated_patterns=36  simulation_time=0.00 sec
```

diagnose_failures fail_patterns -output fail_diags

Example 2

The following example loads patterns from two files into the tool external pattern database and performs two separate diagnoses using the default settings. The example writes the resulting reports to files instead of displaying them onscreen. The failure file, *scan_failure_file*, contains only a scan test section ("scan_test" keyword followed by scan test fails), so the tool performs only a scan diagnosis for that file. The second file, *chain_failure_file*, contains a chain test section ("chain_test" keyword followed by chain test fails) as well as a scan test section. The tool performs only a chain diagnosis for that file.

read_patterns pat_file1
read_patterns pat_file2 -append
diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv
diagnose_failures chain_failure_file -output chain_diag.ascii -replace

Example 3

The following example first performs chain diagnosis on a failure file that contains a chain_test section. The example next attempts a diagnosis of a failure file that contains no chain_test data; however, because the set_diagnosis_options command is set to chain mode, the tool skips this diagnosis.

read_patterns pat_file1
set_diagnosis_options -mode chain
diagnose_failures chain_failure_file -output chain_diag.ascii -replace
diagnose_failures scan_only_failure_file -output scan_diag.ascii -csv scan_diag.csv

```
//  Error: Cannot identify faulty chains. Please either use chain test
//    patterns or specify faulty chains.
```

Example 4

The following example sets the external test pattern source to *pattern_file1*, runs diagnosis on the chain_foo chain failure and the failed patterns failure file.and runs diagnosis on the *chain_foo* chain failure and the *failed cycles* failure file.

set_system_mode analysis
read_patterns pattern_file1
diagnose_failures failed_cycles -faulty_chain chain_foo FAST_TO_FALL

Related Commands

display_diagnosis_report
set_diagnosis_options
set_fault_type
write_flat_model
write_failures

display_specification

Context: dft

Mode: setup, analysis

Usage

display_specification [*id*] [-design_name *design_name*] [-create]

Description

Displays the DftSpecification(design_name,id) configuration data in a tab of the Configuration Tree window of DFTVisualizer.

You can create a DftSpecification from scratch in DFTVisualizer by using the -create option, or also modify an existing DftSpecification.

When only one DftSpecification wrapper is defined in memory, you do not need to supply any arguments; the tool automatically displays the single DftSpecification wrapper.

You define, or modify, DftSpecifications using the Configuration window in DFTVisualizer. This window provides a graphical user interface for inserting Mentor DFT IP into existing designs. For information on using the Configuration window in DFTVisualizer, refer to the section "Generation of a DFT Specification" in the *Tessent IJTAG User's Manual*.

Arguments

- *id*

  An optional string that specifies the DftSpecification wrapper to be displayed. As described in the DftSpecification wrapper syntax, the wrapper is identified by the two strings design_name and id. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(design_name,*), you can omit the id from the display_specification command. However, if several DftSpecification wrappers matching DftSpecification(design_name,*) exist in memory, you must specify the appropriate id when executing the display_specification command. Refer to the description of the -design_name option to understand how the value of design_name is obtained.

- -design_name *design_name*

  An optional switch and string pair that specifies the design_name value used to identify the DftSpecification wrapper. The -design_name value defaults to the current design if it has been set with the set_current_design command. If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

  When the id is specified, if exactly one wrapper matches "DftSpecification (*,<id>)" in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified unless the -create option is specified. When the id is not specified, if exactly one wrapper matches "DftSpecification (*,*)" in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- -create

  An optional switch that disables the existence check of the DftSpecification wrapper and specifies to create and display an empty DftSpecification wrapper. If either of the id or -design_name options are not specified, their values are determined using the heuristic described in the id and -design_name options description. If the tool cannot determine the id, the default is “rtl” in the dft -rtl context and “gate” in the dft -no_rtl context. If the tool cannot determine the design_name, it is set to “?” in the created file which you can then modify in DFTVisualizer. If you do not change the “?” to a valid design name, the tool will generate an error when you attempt to process the DftSpecification.

Example 1

The following example reads a DftSpecification from a file and displays it in DFTVisualizer.

```
set_context dft -rtl
read_config_data ./my_design.dft_spec_rtl
display_specification
```

Example 2

The following example creates an empty DftSpecification and displays it in DFTVisualizer, allowing the user to add its content using the DFTVisualizer graphical user interface. The created DftSpecification wrapper is named “DftSpecification(mydesign,rtl)”. The design_name is taken from the name of the current design. The id defaults to “rtl” because the command was called inside the dft -rtl context.

```
set_context dft -rtl
read_verilog mydesign.v
set_current_design mydesign
display_specification
```

Related Commands

process_dft_specification

display_diagnosis_report

Context: patterns -scan_diagnosis

Mode: analysis

Prerequisites: This command can only operate on a diagnosis report file written out using the diagnose_failures command with the -Output switch.

Usage

display_diagnosis_report ***diagnose_failures_output_filename***

DFTVisualizer Menu Path:
Tools > Diagnosis Report

Description

Opens a diagnosis report in which you can click symptoms and suspect locations to add related gates to the DFTVisualizer Debug window.

The display_diagnosis_report command opens the specified diagnosis report file in File Viewer of DFTVisualizer. In the displayed report, each symptom and all suspect locations (pin and net pathnames) show up as active links. Clicking a suspect's pin_pathname link adds the gate associated with that pin to the DFTVisualizer Debug window. Clicking a suspect's net_pathname link adds all the gates connected to that net. Clicking a symptom link adds all the gates with pins in that symptom's suspect list.

Refer to "Diagnosis Reporting" in the *Tessent Diagnosis User's Manual* for a detailed description of the types of information contained in the diagnosis report.

Arguments

- ***diagnose_failures_output_filename***

  A required string that specifies the pathname of a diagnosis report file written out using the diagnose_failures -Output command.

Example

The following example performs scan diagnosis using the set of ATPG patterns in the pattern file *pat_file1* and the failure information in the failure file *scan_failure_file*, writing the diagnosis report in ASCII format to the file *scan_diag.ascii*. The example then opens DFTVisualizer if it is not already open and opens the diagnosis report in the File Viewer.

```
read_patterns pat_file1
diagnose_failures scan_failure_file -output scan_diag.ascii -replace
display_diagnosis_report scan_diag.ascii
```

Related Commands

diagnose_failures

set_diagnosis_options

display_message

Context: unspecified, all contexts

Mode: all modes

Usage

display_message **string** [-error | -warning | -info | -note]

Description

Sends a message to the specified message stream.

The string can contain \n to break the message into multiple lines. Messages sent in the message stream are prefixed with an "// Error: ", "// Warning:" , "//" , "//" or "//Note:" string as appropriate, and successive lines are automatically indented.

The output of Tcl commands such as **puts** is also redirected to the screen and/or to a log file as with any other tool output.

Arguments

- string

  A required string that specifies the message to be sent to the message stream. The string supports the use of the newline character sequence "\n" to break the message into multiple lines.

- -error

  An optional literal that specifies to send a message into the error stream. This message does not interrupt the execution of the script. You can issue multiple error messages and then terminate the script using a {return -code error "string"} command. This return code causes the dofile to abort unless you have specified set_dofile_abort off.

- -warning

  An optional literal that specifies to display a warning.

- -info

  An optional literal that specifies to display an information note. The message is prefixed by a double slash "//".

- -note

  An optional literal that specifies to display a note.

Example

The following example issues two error messages and then stops the invocation of the current Tcl command using a Tcl return -code error command.

```
display_message "this value is wrong" -error
display_message "this other value is also wrong" -error
return -code error
```

Related Commands

register_tcl_command

unregister_tcl_command

dofile

Context: unspecified, all contexts

Mode: all modes

Usage

dofile ***filename*** [-History]

Description

Executes the commands contained within the specified file.

The dofile command sequentially executes the commands contained in a specified file. This command is especially useful when you must issue a series of commands. Rather than executing each command separately, you can place them into a file in their desired order and then execute them by using the dofile command. You can also place comment lines in the file by starting the line with a double slash (//); the tool handles these lines as comments and ignores them.

The dofile command sends each command expression (in order) to the tool which in turn displays each command line from the file before executing it. If the tool encounters an error due to any command, the dofile command stops its execution, and displays an error message. You can enable the dofile command to continue regardless of errors by setting the set_dofile_abort command to Off.

The dofile command is very similar to the Tcl source command except the dofile command has the following additional usability features:

- Loads relative paths called with dofile <RelativeFilePath> relative to the position of the dofile containing the dofile command.

  You must use the same syntax when using the source command as shown here: source [file dirname [info script]]/<RelativePathToFile>. When the source command is used, the file is processed purely by the Tcl shell.

- Accepts // as comments, as well as #.

- Automatically escapes [#] and [#:#] entries in bus names.

Arguments

- ***filename***

  A required string that specifies the name of the file that contains the commands that you want the tool to execute.

- -History

  An optional switch that specifies for the tool to add the commands from a dofile to the command line history list. By default, the commands in a dofile are not inserted into the history list, but the dofile command itself is added to the list.

Example

The following example executes, in order, all the commands from the file *command_file*:

```
dofile command_file
```

The command_file may contain any Tcl or application command available. An example of a command_file is as follows:

```
foreach_in_collection net [get_nets u1/*] {
    if {[sizeof_collection [get_fanins $net]] == 0} {
        delete_connections [get_fanouts $net -stop_on pin]
        delete_nets $net
    }
}
```

Related Commands

history

set_dofile_abort

echo

Context: unspecified, all contexts

Mode: all modes

Usage

echo ***"string"*** [{> | >>} *file_pathname*]

Description

Issues a user-defined string to the transcript.

The echo command issues a user-defined string to the transcript or to a pathname, if you use one of the file redirection operators.

Note

Commands that use either the > or >> file redirection operator are first checked for correctness. Syntax errors are reported to the display prior to the command's execution. The redirection operator does not hide these errors.

Arguments

- ***string***

  A required string. The string that you want echoed to the transcript. Double quotes are required if the string contains spaces or special characters.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example redirects output from several commands into a single output file, *my_scan_report*. The first command creates or replaces the *my_scan_report* file. The second and following commands append to the same file.

```
echo "----------- scan cells ------------" > my_scan_report
report_scan_cells >> my_scan_report
echo "----------- scan chains ----------" >> my_scan_report
report_scan_chains >> my_scan_report
```

execute_cdp_test

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

Mode: setup, analysis

Usage

execute_cdp_test ***test_name*** [-collect_data_type_list variable]

Description

Executes the specified test using the currently connected tester.

By default, this command returns pass/fail results for the associated test procedure. When you specify -collect_failures variable, the tool collects the failing data during test execution and returns the failing data in a tabulated format. The tool provides the failing data with respect to the ReadVar variables originally specified in the PDL that you used to generate the pattern.

When you are using SimDUT as your tester, the tool converts the test pattern file to simulation events, and then the Verilog simulator returns the failing information to SimDUT.

Refer to "Executing an IJTAG Test Pattern" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a sample usage scenario.

Arguments

- ***test_name***

  A required string that specifies the name of the test you want to execute.

- -collect_data_type_list variable

  An optional switch and literal that specifies to execute the test and return variable failures.

Example 1

The following example executes test1 on the DUT and returns the results for the collect_failures procedure for the test pattern associated with test1.

execute_cdp_test test1 -collect_data_type_list variable

```
//  command: execute_cdp_test test1 -collect_data_type_list variable
//  Test 'test1' failed.
//  Variable failures and unmapped failures of pattern
// 'DLV2_1S0_ETMemory_Full.svf' :
//
//  PatternSet  Variable                        Pin  Expected
//                                                     Actual
//  ----------  ----------------------------  ---  --------
//        main             BP2_WIR.IR_STATUS2  TDO        b1
//                                                        b0
//  ----------  ----------------------------  ---  --------
//        main        BP2_WIR.IR_STATUS1.DONE  TDO        b1
//                                                        b0
//  ----------  ----------------------------  ---  --------
//        main  BP2_WDR.WBP1_INST_POINTER_REG  TDO   b000000
//                                                   b001001
```

```
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_A_ADD_REG_Y  TDO     b0000
//                                                    b1111
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_A_ADD_REG_X  TDO     b0000
//                                                    b1000
//  ----------  ----------------------------  ---  --------
//        main       BP2_WDR.WBP1_GO_ID_REG10  TDO        b0
//                                                       b1
//  ----------  ----------------------------  ---  --------
```

Example 2

The following command returns the pass/fail results for the test pattern associated with test2.

execute_cdp_test test2

```
command: execute_cdp_test test2
Test 'test2' failed.
```

Related Topics

add_cdp_test

launch_sid_tester

execute_gpib_command

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

Mode: setup, analysis

Usage

execute_gpib_command ***command*** **-address** ***gpib_address*** [-read_result [-timeout *read_timeout_period*]]

Description

Note

Using this command assumes that you have configured the equipment on the GPIB as described in General Purpose Interface Bus (GPIB) standard IEEE-488. It also assumes that the GPIB instrument supports the GPIB commands you specify.

Allows you to control and program GPIB equipment.

In Tessent SiliconInsight, you generally use GPIB equipment to control power supplies and clock generators for testing purposes. However, you may also control any GPIB instrument—such as a PMU, temperature control, or digital IO—as long as it supports the GPIB standard.

You must connect the GPIB equipment to a USB-to-GPIB adaptor, which you then connect to the performance board.

Combining this command with the execute_cdp_test and add_cdp_test commands enables you to generate detailed programs for debugging and characterizing DUTs. Refer to "Executing an IJTAG Test Pattern" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a sample usage scenario.

Arguments

- ***command***

  A required string that specifies a GPIB command and arguments. The GPIB command must be a valid GPIB command that is supported by the targetted GPIB instrument. Specify the command between quotes.

- **-address** ***gpib_address***

  A required switch and integer pair that specifies the address of the targeted GPIB equipment, where the *gpib_address* is an integer from 0 to 30. This switch enables execution of the GPIB command on the GPIB equipment configured to listen at the specified address.

- -read_result

  An optional switch that directs Tessent SiliconInsight to read the results that the GPIB equipment returns after it executes the specified GPIB command. Use this switch when the GPIB command is a query command, and you expect a query result from the GPIB

equipment. By default, Tessent SiliconInsight Desktop assumes that the GPIB command is a non-query command, and it does not read data from the GPIB equipment.

- -timeout *read_timeout_period*

  An optional switch and integer pair that specifies the maximum length of time (in milleseconds) to wait for the result of the specified command to appear on the GPIB before timing out. The default wait period is 2 seconds (2000 ms).

Example 1

The following command directs the tool to execute the SetVoltage 1.6V command on the GPIB instrument that is enabled at address 5.

execute_gpib_command “SetVoltage 1.6V” -address 5

Example 2

The following command directs the tool to execute the ReadCurrent command on the GPIB instrument that is enabled at address 12. The ReadCurrent command returns results, which Tessent SiliconInsight Desktop will read because you have specified the -read_result switch. The tool will wait no more than 500 ms for the results to appear.

execute_gpib_command “ReadCurrent” -address 12 -read_result -timeout 500

Related Topics

add_cdp_test

execute_cdp_test

Preparing the Computer for Tessent SiliconInsight Desktop

exit

Context: unspecified, all contexts

Mode: all modes

Usage

exit [-force] [*exitval*]

Description

Terminates the application tool program.

The **exit** command terminates the tool session and returns to the operating system. You should either save the current test patterns or modified design before exiting the tool.

If you are operating in interactive mode (not running a dofile) and you do not save the current test pattern set or modified design, the tool displays a warning message, and you are given the opportunity to continue the session and save the test patterns before exiting.

Arguments

- -force

  An optional argument that forces termination of the current tool session even though unsaved data may be lost.

- *exitval*

  An optional integer that specifies an exit code.

  This option can be used to return a non-zero value when exiting the tool. By default, when there's no error, “0” is returned to the caller while non-zero usually indicates an error condition. The specific value returned can be used to differentiate the type of condition that resulted in issuing the exit command.

Example

The following example quits the tool without saving the current test pattern set.

```
set_system_mode analysis
create_patterns
exit -force
```

expand_compressed_patterns

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

Mode: analysis

Usage

expand_compressed_patterns [-CHAIN ***chain_name***...]
[-MAX_1hot_pattern ***pattern_number***] [-MAP_file ***map_filename***] [-REPlace]

Description

Expands compressed 1-hot patterns for better internal failing flop diagnosis.

See "1hot Compressed Pattern Expansion" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -CHAIN ***chain_name***...

  An optional switch and repeatable string that identifies the scan chain name(s) to process for 1-hot expansion.

- -MAX_1hot_pattern ***pattern_number***

  An optional switch and required integer that specifies the maximum number of patterns to produce during expansion. The default value is 1000. If this limit is exceeded, the tool stops command execution and prints a warning message that the limit was exceeded.

- -MAP_file ***map_filename***

  An optional switch and required string that specifies the map filename where 1-hot pattern-IDs, original pattern-IDs, and mapped channel information is saved. Use the -replace option to overwrite an existing map file.

- -REPlace

  An optional switch for overwriting an existing file.

Related Commands

read_patterns
write_patterns
set_pattern_filtering

extract_icl

Context: all contexts

Mode: all modes

Usage

extract_icl [-write_in_tsdb {on | off | (auto)}] [-design_identifier *string*]

Description

Checks the ICL connectivity rules between Ijtag instances and extracts the top-level ICL module.

An error is generated if an ICL module already exists for the top module (current design).

The command changes the context to patterns -ijtag and goes to the system mode analysis. If ICL extraction is successful, it returns the system mode and the context to what it was when the command was invoked. If ICL extraction fails due to one or more DRC errors, the system mode is left in setup mode and the context is left as patterns -ijtag to enable you to debug the DRC violations. Use the open_visualizer command to open the DFTVisualizer debug environment and debug the DRC violations.

Arguments

- -write_in_tsdb on | off | (auto)

  An optional switch and literal pair that controls whether the extracted ICL is automatically written out in the specified TSDB output directory. The TSDB output directory is specified with the set_tsdb_output_directory command and defaults to “tsdb_outdir”.

  With the default value of “auto”, the ICL file is only written if the directory <tsdb_outdir>/dft_inserted_designs/<current_design>_<design_id>.dft_inserted_design already exists. This directory is typically created by the process_dft_specification command prior to invoking the extract_icl command as shown in Example 1.

- -design_identifier *string*

  An optional switch and value pair that sets the design_identifier. You cannot use this option if you are using extract_icl immediately after the process_dft_specification command because the tool will automatically use the design id of the DftSpecification wrapper. You should use this option only in the rare case where you want to save the extracted ICL file in the TSDB directory, and you have not run the ICL extraction imediately following the DFT step. In this case, the tool will use the design identifier specified or inferred with the “set_context dft” command.

Example 1

The following example shows the use of the extract_icl command directly following the process_dft_specification command.

```
set_context dft -rtl
#commands to load design apprears here

set_current_design CoreA
set_design_level physical_block

set_system_mode analysis

create_dft_specification
process_dft_specification

extract_icl
```

Example 2

The following example shows the use of the extract_icl command as part of the “patterns -scan” context to extract the ICL network such that it can be used for test setup. For efficiency, it is recommended to extract the ICL only once after the DFT is completed, and to reuse the extracted ICL in all pattern -scan usage. ICL extraction may also not be possible after layout if the ICL modules were ungrouped during layout. The standard flow is to use the pre-layout ICL file extracted either on the RTL or netlist view for all subsequent pattern generation phases. Formal verification is typically used to verify that the functional mode, including the IJTAG network, was not affected by the layout process.

```
set_context patterns -scan
#commands to load design and cell library apprears here

set_current_design

extract_icl

set_test_setup_icall myproc1
```

Related Topics

process_dft_specification

open_visualizer

set_system_mode

set_tsdb_output_directory

filter_collection

Context: unspecified, all contexts

Mode: all modes

Usage

filter_collection ***base_collection expression*** [-regexp] [-nocase]

Description

Filters a previously created collection, resulting in a new collection.

In many cases, the commands that create collections support a -filter option, which performs filtering as part of the collection process. Usually this is a more efficient method of filtering. However, you may want to use the filter_collection command if you are filtering a large collection multiple times with different criteria.

The filter_collection command results either in a new collection that contains a subset of the objects in the input base_collection, or in an empty collection, which indicates that the expression filtered out all elements of the input base_collection.

Arguments

- ***base_collection***

  A required string that specifies the base collection to be filtered. This collection is copied to the resulting collection, and objects are removed if they are evaluated as false by the conditional expression.

- ***expression***

  A required string that specifies an expression with which to filter the collection. The attributes used within the equation must exist for the object types contained in the collection. Otherwise, an error is generated. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that specifies that the pattern match is case insensitive.

Example 1

The following example creates a collection of instances of type design.

set var [filter_collection [get_instances *] {type == "design”}]

```
{Adder1 Adder2}
```

Example 2

The following example starts with a collection of all instances and filters the collection based on an attribute_equation of user-defined attributes.

set var [filter_collection [get_instances] {myType1 || myType2 == “ABC"}]

The above example is equivalent to this:

set var [get_instances -filter {myType1 || myType2 == “ABC"}]

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

find_design_names

Context: all contexts

Mode: all modes

Usage

find_design_names ***name_pattern*** [-LOcal | -Hier]
[-Design | -NETList | -LIbrary | -All]
[-INStance | -Net | -Pin [INPut | OUtput | INOut | ALLIn | ALLOut] |
[[-Cell | -Module] -LISt_references]]
[{> | >>} *file_pathname*]

Description

Displays design object hierarchical names matched by an input string, which may include asterisk (*) or question mark (?) wildcard characters.

Arguments

- ***name_pattern***

  A required string, which may include asterisk (*) or question mark (?) wildcard characters.

- -LOcal

  An optional switch that matches wildcard characters within the current hierarchy level.

- -Hier

  An optional switch that matches the regular expression across hierarchy boundaries (for example, a* matches a1/b/c). This is the default.

- -Design

  An optional switch that matches only pathnames to objects at the topmost library cell level.

- -NETList

  An optional switch that matches objects from the top of the design down to the topmost library cell.

- -LIbrary

  An optional switch that matches objects within any level of library cells.

- -All

  An optional switch that specifies matches objects at all levels of the design. This is the default.

- -INStance

  An optional switch that matches only instance pathnames. This is the default.

- -Net

  An optional switch that matches only net pathnames.

- -Pin

  An optional switch that matches only pin pathnames (any pin direction). The following optional pin filters restrict which pins are matched:

  INPut — Match only input pin pathnames.

  OUtput — Match only output pin pathnames.

  INOut — Match only bidirectional pin pathnames.

  ALLIn — Match both input and bidirectional pin pathnames.

  ALLOut — Match both output and bidirectional pin pathnames.

- -Cell

  An optional switch that finds all library cell (model) names matching the specified regular expression.

- -Module

  An optional switch that finds all netlist module names matching the specified regular expression.

- -LISt_references

  An optional switch for use with -Cell or -Module that lists the pathnames of all instances in the current design that reference each cell or module matching the regular_expression.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following examples display object pathnames for various input wildcard strings, given a netlist with the following instance hierarchy:

```
/
   tiny_i
      U5
   ret_i
      intreg1_reg_0 ... intreg1_reg_31
      add_20
         U1_0 ... U1_3
      add_30
         U5 ... U12
      mul_18
         U5 ... U868
         FS
            U5 ... U33
      mul_19
         FS
```

```
            U5 ... U278
         U5 ... U181
      mul_22
         U5 ... U735
         FS
            U15 ...
```

and assuming the U5 instances all reference the following library cell:

```
model LSR2BUFA(Q, QN, S, R, G, SD, RD) (
   input(S, R, G, SD, RD) ()
   output(Q) (primitive = _buf UP1 (QT, Q);)
   output(QN) (primitive = _buf UP2 (QNT, QN);)
   intern(QT_int) (instance = LSI_LSR2 UD1 (QT_int, S, R, G, SD, RD);)
   intern(QNT_int) (instance = LSI_LSR2N UD2 (QNT_int, S, R, G, SD, RD);)
   intern(QT) (instance = LSI_NOTI UD3 (QT, QT_int);)
   intern(QNT) (instance = LSI_NOTI UD4 (QNT, QNT_int);)
)
```

Example 1

find_design_names /ret_i/add_2* -instance -design -hier

```
// Note: Matched 4 names
/ret_i/add_20/U1_0
/ret_i/add_20/U1_1
/ret_i/add_20/U1_2
/ret_i/add_20/U1_3
```

Example 2

find_design_names /ret_i/add_2* -instance -netlist -hier

```
// Note: Matched 5 names
/ret_i/add_20
/ret_i/add_20/U1_0
/ret_i/add_20/U1_1
/ret_i/add_20/U1_2
/ret_i/add_20/U1_3
```

Finds instance add_20 under /ret_i/, and also descends the hierarchy to find all netlist instances under /ret_i/add_20/.

Example 3

find_design_names /ret_i/add_2* -inst -netlist -local

```
// Note: Matched 1 names
/ret_i/add_20
```

This example shows that -Local does not descend the hierarchy to find more matches as the previous example does.

Example 4

find_design_names /ret_i/add_2* -ins -design -local

```
// Note: Matched 0 names
```

There are no instances of a library cell under /ret_i/ with instance name starting with add_2.

Example 5

find_design_names /ret_i/*_2? -ins -netlist -local

```
// Note: Matched 2 names
/ret_i/add_20
/ret_i/mul_22
```

Found 2 instances under /ret_i/.

Note

/ret_i/gt_68_2 did not match because the “?” in the wildcard expression requires another character after the “_2”.

Example 6

find_design_names */U5 -inst -design -hier

```
// Note: Matched 7 names
/tiny_i/U5
/ret_i/add_20/U5
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 7

find_design_names ret_i/mul*/U5 -ins -des -local

```
// Note: Matched 3 names
/ret_i/mul_18/U5
/ret_i/mul_19/U5
/ret_i/mul_22/U5
```

Example 8

find_design_names ret_i/mul*/U5 -ins -design -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/FS/U5
/ret_i/mul_19/U5
/ret_i/mul_19/FS/U5
/ret_i/mul_22/U5
```

Example 9

find_design_names ret_i/mul_18/U5* -ins -library -hier

```
// Note: Matched 5 names
/ret_i/mul_18/U5
/ret_i/mul_18/U5/UD1
/ret_i/mul_18/U5/UD2
/ret_i/mul_18/U5/UD3
/ret_i/mul_18/U5/UD4
```

Example 10

find_design_names ret_i/mul*/U5/* -pin output -design -local

```
// Note: Matched 6 names
/ret_i/mul_18/U5/Q
/ret_i/mul_18/U5/QN
/ret_i/mul_19/U5/Q
/ret_i/mul_19/U5/QN
/ret_i/mul_22/U5/Q
/ret_i/mul_22/U5/QN
```

Example 11

find_design_names * -mod -list_references

```
// Note: Found 3 netlist modules
// Note: Instances of netlist module or_gate
/o1
/o2
// Note: Instances of netlist module pme_caam_RC_CG_MOD_335839
/RC_CG_HIER_INST3921
```

Example 12

find_design_names * -cell -list

```
// Note: Found 6 library cells
// Note: Instances of library cell HD45_SZ_MUX21X2
/mux
// Note: Instances of library cell HD45_SZ_LDLRX2
// Note: Instances of library cell HD45_SH_OR2X2
/o1/foo
/o2/foo
// Note: Instances of library cell HD45_SV_CNHLSX4
/RC_CG_HIER_INST3921/RC_CGIC_INST
// NOte: Instances of library cell HD45_SH_SDFPRQX4
/data_out_reg_0
/data_out_reg_1
/data_out_reg_2
/data_out_reg_3
/data_out_reg_4
/data_out_reg_5
/shadow
/non_scan
```

foreach_in_collection

Context: unspecified, all contexts

Mode: all modes

Usage

foreach_in_collection ***itr_var collection* {*body*}**

Description

Iterates over each element in a collection.

You cannot use the Tcl-supplied foreach command to iterate over collections because foreach operates on a Tcl list, and a collection is not a Tcl list but a pointer to an array of data on the C++ side. You can use the get_name_list command to convert the collection of objects into a Tcl list of object names, but you should avoid doing this if the collection is very large or if the object names are not unique, such as port names. For example, a port object is a specific port name on a specific module. Specifying just the name of the port does not indicate which module it belongs to.

You can nest the foreach_in_collection command within other control structures, including foreach_in_collection.

A collection remains in memory for as long as it is referenced by a Tcl variable. A collection is deleted once it is no longer referenced by any Tcl variable. If you make the mistake of using the Tcl command **foreach** to iterate on a collection, the command has the negative side effect of de-referencing the collection and may destroy it. Always use foreach_in_collection to iterate on the elements of a collection.

Arguments

- ***itr_var***

  A required string that specifies the name of a Tcl variable. During each iteration, itr_var holds a string pointer to a collection of exactly one object.

- ***collection***

  A required value that specifies a collection over which to iterate. It may be the content of a Tcl variable that points to a collection or the result of an introspection command that generates a collection. Note, just like the Tcl foreach command, a snapshot of the collection is taken before the first iteration. This means that even if the append_to_collection command is used to increase the size of the collection as part of the iterations, the foreach_in_collection command will not see the newly added elements. Example 3 shows how to collect new elements into a separate collection and use a Tcl while command to repeat the loop until the collection is truly empty.

- ***body***

  A required string that specifies a script to execute per iteration. The entire body must be enclosed within a set of braces.

Example 1

The following example connects a set of pins that has a special attribute called myAtt set to True to a net that has the same name as the pin. If not already present, the net is created. Note that the example assumes that all pins are scalar objects. Extra code would be needed to deal with bused pins and create the entire bused nets with one command. The is_bus and other built-in bus attributes are available to do this efficiently. Notice how the lindex command converts the return value of the get_attribute_value_list command into a string so that it can be used to construct the net name using ${parent}/${leaf}.

```
foreach_in_collection pin [get_pins -filter "myAtt"] {
        set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]
        set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]
        if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]] = 0} {
            create_net ${parent}/${leaf}
        }
        create_connections $pin ${parent}/${leaf}
}
```

Notice how filtering based on the value of an attribute is performed to reduce the size of the collection on which to iterate. The user could have chosen to loop on all pins and check inside the loop if the myAtt attribute is set to true, but this is very inefficient because filtering is done in C++ as opposed to looping in Tcl.

```
#inefficient use of looping. Best to filter collection before looping on
it as shown above.

foreach_in_collection pin [get_pins] {
        if {[get_attribute_value_list $pin -name myAtt]} {
            set leaf [lindex [get_attribute_value_list $pin -name leaf_name] 0]
            set parent [lindex [get_attribute_value_list $pin -name parent_instance] 0]
            if {[sizeof_collection [get_nets ${parent}/${leaf} -silent]] = 0} {
                create_net ${parent}/${leaf}
            }
            create_connections $pin ${parent}/${leaf}
        }
}
```

Example 2

The following example loops on a collection of pins and deletes them. The same pin object is added twice into the collection being looped on to illustrate the fact that obsolete objects are still looped on and that a -silent switch must be used so that they are ignored by the delete_pin command. Without the -silent option, an error would be generated the second time the pin object u1/P2 was looped on.

```
set my_pins [get_pins {u1/P2 u1/P*}]
puts "my_pins are [get_name_list $my_pins]"
foreach_in_collection pin $my_pins {
        delete_pin $pin -silent
}
```

```
my pins are u1/P2 u1/P1 u1/P2 u1/P3
```

Example 3

The following example shows how to iterate on a collection of objects until no objects are left to process. The procedure called “proc_that_returns_new_objects” is called on each object and may return new objects to process. The “append_to_collection col” command cannot be used to process the new objects because the foreach_in_collection command took a snapshot of “col” before the first iteration. Instead, the new objects are collected into a collection variable called “newCol” and it is assigned back to “col” after the foreach_in_collection command is done. The loop is repeated for as long as “newCol” is not empty after the foreach_in_collection command has completed.

```
set col [<start collection>]
while {[sizeof_collection $col] > 0} {
  set newCol {}
  foreach_in_collection obj $col {
    append_to_collection newCol [proc_that_returns_new_objects $obj]
  }
  set col $newCol
}
```

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
filter_collection
get_icl_pins
get_common_parent_instance
get_modules
get_nets
index_collection
is_collection
remove_from_collection
sizeof_collection
sort_collection

get_attribute_list

Context: unspecified, all contexts

Mode: all modes

Usage

Usage 1: List attributes specified on a group of objects

get_attribute_list ***obj_spec*** [-predefined | -user_defined] [-silent]

Usage 2: List attributes registered for a groups of object types

get_attribute_list **-object_types *type_list*** [-predefined | -user_defined] [-silent]

Usage 3: List all registered attributes

get_attribute_list [-predefined | -user_defined] [-silent]

Description

Returns an alphabetically sorted Tcl list of attribute names.

Usage 1: Returns the attributes set on any object found in *obj_spec*. The attributes on an object inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object are reported in the list of attributes for the instances of that module. By default, attributes that have not been set on the objects are not listed unless the attribute was registered with the -show_default option.

If an object in *obj_spec* does not exist, an error message displays unless the -silent switch is specified.

Usage 2: Returns the attributes registered for all specified object types. These do not include the inherited attributes from one object type to another such as the module attribute inherited by the associated instances.

Usage 3: Returns all registered attributes

In all three usages, only predefined attributes are returned when the -predefined option is specified. Only user-defined attributes are returned when the -user_defined option is specified.

Arguments

- ***obj_spec***

  A required value used in Usage 1 that specifies an object name, a Tcl list of object names, or a collection of objects.

- **-object_types *type_list***

  A required switch and value pair used in Usage 2 that enumerates the object types for which the attribute list is requested. The currently supported object types are module, port, instance, pin, and net. The list of supported object types will be augmented in successive releases and eventually all object types with be available for introspection.

- -predefined | -user_defined

  An optional switch that constrains the returned list to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default no constraints are applied and both pre-defined and user-defined attributes are returned.

- -silent

  An optional switch that specifies to suppress error messages if there are objects in the *object_spec* that do not exist.

Example 1

The following example lists the attributes set on three pins:

get_attribute_list {tiny1_3/d1 tiny1_3/d2 tiny1_3/d3}

```
base_name direction has_functional_source is_hard_module is_valid
leaf_name library_name master_module_name module_name name object_type
parent_instance parent_is_hard_module tie_value
```

Example 2

The following example shows the impact of using the -show_default switch, when registering an attribute, on the get_attribute_list command:

register_attribute -name a1 -value_type boolean -object_types pin
get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
{}
```

register_attribute -name a2 -value_type boolean -object_types pins -show_default
get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
a2
```

set_attribute_value u1/pin1 -name a1

```
{u1/pin1}
```

get_attribute_list {u1/pin1 u1/pin2} -user_defined

```
a1 a2
```

Related Commands

get_attribute_value_list
register_attribute
report_attributes
reset_attribute_value
set_attribute_options
set_attribute_value
unregister_attribute

get_attribute_option

Context: unspecified, all contexts

Mode: all modes

Usage

get_attribute_option **-name** ***attribute_name*** **-object_type** ***type_name***
{[-export_during_write {auto | off | on}] |
[-preserve_boundary_in_flat_model] |
[-applies_to_child_instances] |
[-display_in_gui **{**off **|** on**}] |**
[-gui_marking_index *marking_index*]}

Description

Retrieves the current setting of an attribute's option.

The get_attribute_option command returns a string with the option settings of the specified attribute. These options can be modified after the attribute has been registered using the set_attribute_options command.

Arguments

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute whose option values are to be returned.

- **-object_type** ***obj_type***

  A required switch and value pair that specifies the object type whose attribute option settings are to be returned.

- -export_during_write

  An optional switch that retrieves the current value of the export_during_write option for the specified attribute name on the specified object type. For more information on the -export_during_write switch, see the set_attribute_options command.

- -preserve_boundary_in_flat_model

  An optional switch that retrieves the current value of the -preserve_boundary_in_flat_model option for the specified attribute name on the specified object type. For more information on the -preserve_boundary_in_flat_model switch, see the set_attribute_options command.

- -applies_to_child_instances

  An optional switch that retrieves the current value of -applies_to_child_instances for the specified attribute name on the specified object type. For more information on the -applies_to_child_instances switch, see the set_attribute_options command.

- -display_in_gui

  An optional switch that retrieves the current value of the -display_in_gui option for the specified attribute name on the specified object type. For more information on the -display_in_gui switch, see the set_attribute_options command.

- -gui_marking_index

  An optional switch that retrieves the current value of the -gui_marking_index for the specified attribute name on the specified object type. For more information on the -gui_marking_index switch, see the set_attribute_options command.

Example

The following example retrieves the -applies_to_child_instances setting for the attribute named my_power_domain which is registered for the instance object type:

```
get_attribute_option -name my_power_domain -object_type instance
        -applies_to_child_instances
```

```
on
```

Related Commands

get_attribute_value_list
register_attribute
report_attributes
reset_attribute_value
set_attribute_options
set_attribute_value
unregister_attribute

get_attribute_value_list

Context: unspecified, all contexts

Mode: all modes

Usage

get_attribute_value_list ***object_spec*** **-name** ***attribute_name*** [-silent]

Description

Retrieves the value of an attribute on the specified design objects.

Like all other Tessent commands ending with an _list suffix, this command returns a list with all the values set for the attribute on each of the objects in object_spec.

If an object in object_spec does not exist, or if the attribute specified by attribute_name does not exist on the design object, an error message displays unless the -silent switch is specified.

Arguments

- ***object_spec***

  A required value that specifies an object name, a Tcl list of object names or a collection of objects.

- **-name** ***attribute_name***

  A required switch and string pair that specify the name of the attribute whose value is to be returned.

- -silent

  An optional switch that specifies to suppress error messages if an object in object_spec does not exist, or if the attribute specified by attribute_name does not exist for a given design object.

Example 1

The following example returns the value of parent_instance value for pins. The get_attribute_value_list command returns a Tcl list of values even if there is only one object in obj_spec. Use lindex as shown below to convert any one element list into a string that you can then concatenate with other strings to create object names. The 0 argument to the lindex command below takes the first (and only) element of the list.

get_attribute_value_list u1/u5/u8/ABC -name parent_instance

```
u1/u5
```

get_attribute_value_list u1/DEF -name parent_instance

```
{}
```

lindex [get_attribute_value_list u1/DEF -name parent_instance] 0

Example 2

The following example retrieves values for the attribute named “is_hard_module” on the listed modules.

get_attribute_value_list {alu mux2 rom3} -name is_hard_module

```
false true false
```

set myList {mux2 alu}
get_attribute_value_list $myList -name is_hard_module

```
true false
```

Related Commands

get_attribute_option
get_attribute_value_list
register_attribute
report_attributes
reset_attribute_value
set_attribute_options
set_attribute_value
unregister_attribute

get_cdp_test_list

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

Mode: setup, analysis

Usage

get_cdp_test_list

Description

Returns a Tcl list of tests currently available in the CDP.

Example

The following example shows how you can use this command to return a Tcl list of the current tests within the CDP.

```
foreach test [get_cdp_test_list] {
        execute_cdp_test $test
}
```

```
taptest membist1 membist2 logicbist1 logicbist2
```

Related Topics

execute_cdp_test

get_clock_list

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Usage

get_clock_list

Description

Returns a list of all clocks specified with the add_clocks command.

Arguments

None.

Example

The following example adds two clocks to the clock list and then returns a list of the clocks:

```
add_clocks 1 clk1
add_clocks 0 clk0
get_clock_list
clk1 clk0
```

Related Commands

add_clocks
delete_clocks
get_clock_option
report_clocks

get_clock_option

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Usage

get_clock_option ***pin_pathname*** { -is_sync | -off_state | -free_running
| -period [-timeplate *timeplate*] [-time_unit { ps | ns | us | ms | s }]
| -divider | -always_capture | -internal }

Description

Returns the setting of an option specified with the add_clocks command.

The get_clock_option command returns the setting of a single option type for one or more defined clocks. You can use the get_clock_list command to gather the list of defined clocks.

Arguments

- ***pin_pathname***

  A required value that specifies one or more pin names previously defined as a clock with the add_clocks command.

  The get_clock_option command requires one of the following switches:

- -is_sync

  An optional switch that returns a ‘1’ if driven from a timeplate and a ‘0’ if specified by the “add_clocks -period” command.

- -off_state

  An optional switch that returns the value of the off-state (1 or 0).

- -free_running

  An optional switch that returns a ‘1’ if the pin_pathname is a free-running clock, and returns a ‘0’ if not.

- -period [-timeplate *timeplate*] [-time_unit { ps | ns | us | ms | s }]

  An optional switch that returns the clock period if previously specified with the “add_clocks -period” command. Otherwise, you must use the -timeplate switch to identify the timeplate that drives the clock. You can use the -time_unit switch if you want to return the clock period in units other than nanoseconds.

- -divider

  An optional switch that returns an integer value that specifies the divider of a free-running clock, which you can set with the “add_clocks -divider” command. The switch returns an integer value of ‘1’ for undivided or non-free-running clocks.

- -always_capture

  An optional switch that returns a ‘1’ if the pin is an always-capture clock, and returns a ‘0’ if not.

- -internal

  An optional switch that returns a ‘1’ if the clock was previously specified with the “add_clocks -internal” switch, and returns a ‘0’ if not.

Example

The following example adds two clocks to the clock list, reports on the clocks, then shows two ways of using the get_clock_option command:

SETUP> **add_clocks 1 I8**
SETUP> **add_clocks 0 I6 -free_running -period 5**
SETUP> **report_clocks**

```
User-defined Clocks (2):
 ---------  ------------  --------  ----  ----------------
 Off State   Constraints  Internal  Name  Other Properties
 ---------  ------------  --------  ----  ----------------
     1                       No     I8
     0      Asynchronous     No     I6    (period:5.00ns)
```

SETUP> **get_clock_option {I6 I8} -off_state**
0 1

SETUP> **get_clock_option [get_clock_list] -free_running**
1 0

Related Commands

add_clocks
delete_clocks
get_clock_list
report_clocks

get_common_parent_instance

Context: all contexts

Mode: all modes

Usage

get_common_parent_instance ***object_spec***

Description

Returns the common ancestor of all instances, pins, or nets found in object_spec.

You can use this command during dft insertion when you want to place a driving gate at the common ancestor of all its fanouts.

Arguments

- ***object_spec***

  A required value that specifies an object name, a Tcl list of object names, or a collection of objects to find their common ancestor.

Example

The following example finds the common ancestor of all pins having the attribute myAtt set to true. The name of the returned instance object is then used to instantiate a driver cell. Notice how quotes are used around the concatenation of ${parent_name} and /myGate to make sure that any white spaces that may exist in ${parent_name} (when it contains escaped identifiers) are preserved.

```
set parent [get_common_parent_instance [get_pins -filter myAtt]]
set parent_name [get_single_name $parent]
set driver [create_instance “${parent_name}/myGate” -of_module MyMod]
```

Related Commands

create_instance
delete_instances
rename_instance
replace_instances
uniquify_instances

get_config_elements

Context: unspecified, dft, patterns

Mode: all modes

Usage

get_config_elements [*name_patterns*] [-hierarchical] [-in_wrappers *wrapper_object_spec*]
[-partition *partition*] [-count] [-type *type*] [-filter *filter*]
[-regexp] [-nocase] [-silent]

Description

Returns a collection of configuration elements or a count of configuration elements when the -count option is used.

When no name_patterns is specified, then all elements are returned unless the -in_wrapper option is specified, in which case only the elements in the specified wrappers are returned. When using one or many name_patterns, only the elements whose name matches one of the name_patterns are returned.

Arguments

- *name_patterns*

  An optional string that specifies one name pattern or a Tcl list of name patterns to search. The name patterns are automatically decomposed into a series of anchored leaf name patterns using the / as a separator. The name identifiers are always matches ignoring case. The ids found inside the parentheses are matched taking into account case unless the -nocase option is used. The comma has special meaning inside the parentheses and is used to split the patterns and match each id independently. A leaf pattern that does not include a set of parentheses matches all elements matching the leaf name irrespective of whether it has ids or not. See the examples below for a complete illustration of the matching method.

  When the -in_wrapper option is specified, the name_patterns are described relative to the specified wrappers, When the -hierarchical option is specified, the name_patterns are matched recursively against all elements below the specified wrappers.

- -hierarchical

  An optional string that specifies that the name_patterns are matched recursively against all elements below the specified wrappers.

- -in_wrapper *wrapper_object_spec*

  An option value pair that specifies that the configuration elements are to be searched inside specific wrappers. When the -in_wrapper option specified, the name_patterns are searched relative to the specified wrappers. The wrapper_object_spec can be the complete names of wrappers relative to the root of the partition or a collection containing one or many wrapper elements as returned by get_config_elements or “get_config_value -object”. When specifying the names of the wrapper, you can use the -partition option to specify in which partition the wrappers exist.

- -partition *partition_name*

  An optional option value pair that specifies the partition from which the configuration element is to be obtained. When using the -in_wrapper option, the -partition option can only be specified if the wrapper_object_spec is the name of a wrapper. When the wrapper_object_spec is a collection containing wrapper objects, the -partition option cannot be specified because a specific partition is already associated to the configuration objects. All configuration data you normally interact with is in the default partition, so you typically never need to specify the -partition option.

  The tool currently defines two partitions: the default partition in which all user specifications are stored, and the tcd partition which is a read-only partition in which the read_core_descriptions commands stores the data contained inside the Core(module_name) wrapper. The default partition contains the definition of DftSpecification, and DefaultsSpecification/DftSpecification wrappers.

- -count

  An optional Boolean option that specifies to return the number of elements matched as opposed to the elements themselves.

- -type wrapper | data_wrapper | csv_wrapper | property

  An optional option value pair that specifies to limit the search to a specific configuration element type. When the -type option is omitted, all types are considered in the matching.

- -filter attribute_equation

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the configuration element types being filtered. See section "Attribute Filtering Equation Syntax" for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when matching the Ids of wrapper and repeatable properties. The wrapper and property names are always matched ignoring casing.

- -silent

  An optional option that suppress the warning that is normally generated when no configuration element matched the name_patterns.

Example

The following example operates on the following configuration data:

```
tmp(1) {
  ABC (Def,2) {
    prop1 : 1;
    Prop2 : 2;
  }
  abc(def,1) {
  }
  abcdef {
  }

}
```

Case 1

This matches all wrappers with a leaf name starting with abc inside wrapper tmp(1) independent of whether it has an id. The matching is case-insensitive because only the id matching is case-sensitive.

```
> get_config_element abc -in tmp(1)
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 2

This matches the wrappers with the first id equal to Def. Notice that it does not match abc(def,1) as ids are matched considering casing unless the -nocase option is used.

```
> get_config_elements *(Def,*) -in tmp(1)
{/tmp(1)/ABC(Def,2)}

> get_config_elements *(Def,*) -in tmp(1) -nocase
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}
```

Case 3

This matches elements below the top-level wrapper starting with a*.

```
> get_config_element */a*
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1) /tmp(1)/abcdef}}
```

Case 4

This matches elements below the top-level wrapper starting with a* and having two ids.

```
> get_config_element */a*(*,*)
{/tmp(1)/ABC(Def,2) /tmp(1)/abc(def,1)}}
```

Case 5

This matches elements starting with ‘p’ anywhere inside tmp(1).

```
> get_config_element p* -in tmp(1)  -hierarchical
{/tmp(1)/ABC(DeF,2)/prop1 /tmp(1)/ABC(DeF,2)/Prop2}
```

Case 6

This matches elements starting with ‘p’ anywhere inside tmp(1) and returns the amount of elements that matched instead of the elements themselves.

```
> get_config_element p* -in tmp(1)  -hierarchical -count
2
```

Related Commands

add_config_element

get_config_value

get_config_messages

Context: unspecified, dft, patterns

Mode: all modes

Usage

get_config_messages *config_object_spec*
[-string | {-objects [-hierarchical]}]
[-all | {[-error] [-warning] [-info]]

Description

Returns a list of message strings attached to a configuration element.

By default, all message types are returned unless the -error, -warning, and/or -info options are specified. If the -objects option is specified, the command returns the configuration element if it has messages of the specified type.

Arguments

- *config_object_spec*

  A required value that specifies the name of a configuration element or a collection containing one configuration element.

- -string

  An optional Boolean option that specifies that the command is to return the messages formatted into strings exactly as they are shown with the report_config_messages command. The entries start with E#), W#) and I#) based on the type of the message and # is incremented from 0 independently for each type.

  The use of -string is mutually exclusive with the -objects option, When either is specified, -string is assumed.

- -objects

  An optional Boolean option that specifies to return a collection of configuration elements having at least one message of the specified type.

- -hierarchical

  An optional Boolean option that specifies that the config_object_spec is a wrapper and that the command should return all elements below the specified wrapper having at least one configuration message for the specified types. This option is only allowed in combination with the -objects option

- -all

  An optional Boolean option that specifies that all messages types are to be returned. The use of the -all option is mutually exclusive with the specification of the -error, -warning and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages returned are of type warning. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning, and -info options is specified, -all is assumed.

- -info

  An optional Boolean option that specifies that the messages returned are of type info. The use of the -error option is mutually exclusive with the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

Example

The following example adds an error message to the property rtl_extension inside DftSpecification(mydesign,rtl). The get_config_messages is then used to get the message string list on the property. The command “get_config_messages -objects -hierarchical” is also used to get a collection of all elements below DftSpecification(mydesign,rtl) with an error message.

```
> set_context dft -rtl
> set spec [add_config_el DftSpecification(mydesign,rtl)]
> add_config_message "this is an error string \non two lines" \
       -config_element [get_config_el rtl_extension -in $spec] \
       -error -display
//  Error: /DftSpecification(mydesign,rtl)/rtl_extension
//         this is an error string
//         on two lines
> get_config_messages [get_config_el rtl_extension -in $spec]
E0) this is an error string
    on two lines
> get_config_messages $spec -objects -hier
{/DftSpecification(mydesign,rtl)/rtl_extension}
```

Related Commands

add_config_message
delete_config_messages
report_config_messages

get_config_selected_nodes

Context: dft, patterns

Mode: all modes

Usage

get_config_selected_nodes

Description

Returns a collection containing the configuration elements corresponding to the tree nodes currently selected in the Configuration Data window.

Arguments

None

Example

The following example shows the result of the command when two nodes are selected in the Configuration Data window as shown in Figure 7-11. You can select more than one element by pressing the Shift or Control key while selecting the second element.

```
> get_config_select_node
{/DftSpecification(mydesign,rtl)/MemoryBist/Controller(1)
/DftSpecification(mydesign,rtl)/MemoryBist/Controller(2)}
```

Figure 7-11. Selecting Two Nodes in the Configuration Tree

Related Commands

display_specification

add_config_tab

get_config_value

Context: unspecified, all contexts

Mode: all modes

Usage

get_config_value *config_object* [-in_wrapper *wrapper_object_or_name*]
[-time_unit {s | ms | us | ns | ps | fs}]
[-number_format {int | bin | hex}]
[-reference_name | -reference_object | -parent_name | -parent_object | -root_name | -type | -name | -id *id* | -value_id *value_id* | -is_specified | -id_count | -value_count | -exists | -root_name | -root_object | -leaf_name | -base_name |-meta_name | -meta_object | -meta_id | -partition_name]
[-default_value] [-tolower | -toupper] [-file_name] [-object]
[-partition *partition*] [-silent]

| *id_name* | all} | -value_id {*<position>* | *value_name* | all} |*value_name* | all} |

Description

Returns different values associated with a configuration element based on the specified option.

Figure 7-12 below illustrates some of those values.

Figure 7-12. Various Components of a Configuration Element

Arguments

- ***config_spec***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When config_spec is a collection, the -in_wrapper and -partition options cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the config_spec value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)” and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- -in_wrapper *wrapper_object_spec*

  An optional option value pair that is used to specify the parent wrapper in which the configuration element exists. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The name in the config_element_name is relative to the specified parent wrapper.

- -time_unit s | ms | us | ns | ps | fs | as_is

  An optional Boolean option that can be used when the command is returning a Value time property value; it is ignored otherwise. When specified or defaulted to as_is, the time value is returned unmodified with the unit string specified if it existed in the value of the configuration element. When specified to a value other than as_is, the time value is converted into a real number formatted to the specified units. The unit string is also dropped.

- -number_format int | bin | hex | as_is

  An option value pair that can be used when the command is returning a Value binary property or hexadecimal value. It is ignored otherwise. When specified or defaulted to as_is, the value is returned unmodified with the format if it existed in the value of the configuration element. When specified to a value other than as_is, the binary or hexadecimal value is converted into the specified format.

- -reference_name

  An optional Boolean option that specifies that the data requested is the name of the referenced metadata element. If the metadata is not a Wrapper reference property, this returns a null string.

- -reference_object

  An optional Boolean option that specifies that the data requested is a collection containing the referenced metadata element. If the metadata is not a Wrapper reference property, this returns an empty collection.

- -parent_name

  An optional Boolean option that specifies that the data requested is the name of the wrapper in which the configuration element exists. If the element is a root wrapper, the string / is returned.

- -parent_object

  An optional Boolean option that specifies that the data requested is a collection containing the wrapper in which the configuration element exists. If the element is a root wrapper inside a given partition, a collection containing the partition in which the root wrapper exists is returned.

- -type

  An optional Boolean option that specifies that the data requested is the type of the configuration element. The type is either wrapper, data_wrapper, csv_wrapper, property,

repeatable_property, data_element, or csv_element. Table 7-9 explains each type in more details.

Table 7-9. Types of Configuration Elements

Type	Description
wrapper	Type returned for a Wrapper used to groups other wrappers and properties.
data_wrapper	Type returned for a DataWrapper used to group data_row elements.
csv_wrapper	Type returned for a CsvWrapper used to group csv_row elements.
property	Type returned for a Property.
repeatable_property	Type returned for a RepeatableProperty property.
data_row	Type returned for elements found inside a DataWrapper.
csv_row	Type returned for elements found inside a CsvWrapper.

- -name

  An optional Boolean option that specifies that the data requested is the name of the configuration element. As shown in Figure 7-12, the name is the complete name including all hierarchical levels. If at any level in the hierarchy, the leaf_name is not unique, the leaf_name is replaced by a positional index so as to have a unique name for each element. For example, if there are two instances of the same wrapper called “port(abc)” inside the same parent wrapper called “ports”, the name would be reported as “/Ports/Port(<2>)” assuming it is the third element inside “/ports”.

- -id *position* | *id_name* | all

  An optional option value pair that specifies that the data requested is the id of a wrapper or of a repeatable property. When using -id all, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non repeatable Id declared for it. In this case the value is returned as a tcl string which is the case for most wrappers and repeatable properties you will introspect.

  When you have configuration elements with more than one Id, you can query one id at a time using either its position or when it has metadata, the id of the ID wrapper that declares the id. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on. In such a case, the id is returned as a string except when using the id of a repeatable id in which case a well-formatted Tcl list is returned.

- -value_id *position* | *value_name* | all

  An optional option value pair that specifies that the data requested is the value of a property, repeatable property, or of a row inside a Data or CSV wrapper. If none of the options listed between -value_count and -reference_object inclusively are specified, “-value_id all” is assumed.

When using “-value_id all”, the command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the value is returned as a Tcl string which is the case for most properties you introspect.

When you have configuration elements with more than one elements, you can query one value at a time using either its position or, if it has metadata, the id of the Value wrapper that declares the value. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on. In such a case, the value is returned as a string except when using the id of a repeatable Value in which case a well-formatted Tcl list is returned.

- -is_specified

  An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element is specified and 0 when it is not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and regardless of whether the singular wrapper or property is specified. The parent wrapper must exist before you can query if one of its element is specified. Unspecified elements are not included in the output file when using the write_config_data or the report_config_data commands unless the -show_unspecified option is used.

- -id_count

  An optional Boolean option that specifies that the data requested is the number of ids specified for the wrapper of repeatable property. A value of 0 is returned for configuration objects that do not have ids like a wrapper declared without any Id wrapper or a property.

- -value_count

  An optional Boolean option that specifies that the data requested is the number of values specified for the property, the Data or CSV wrapper row. A value of 0 is returned for configuration objects that do not have values like a wrapper. Implicit Boolean properties still report one value because the true or false value is implicit but still present.

- -exists

  An optional Boolean option that specifies that the data requested is a Boolean value that is 1 when the configuration element exists and 0 when it does not. Remember that singular wrappers and properties exist as soon as their parent wrapper exists and whether the singular wrapper or property is specified or not. You use the -is_specified option to know if such elements are specified or not.

- -root_name

  An optional Boolean option that specifies that the data requested is the name of the root wrapper in which the configuration element exists. If the element is a root wrapper, its name is returned.

- -root_object

  An optional Boolean option that specifies that the data requested is a collection containing a root wrapper in which the configuration element exists. If the element is a root wrapper, it is returned in the collection.

- -leaf_name

  An optional Boolean option that specifies that the data requested is the leaf name of the configuration element. As shown in Figure 7-12, the leaf name is the last level of hierarchy including the base_name and the ids when they exist.

- -base_name

  An optional Boolean option that specifies that the data requested is the base name of the configuration element. As shown in Figure 7-12, the base name is the last level of hierarchy excluding the ids when they exist.

- -meta_name

  An optional Boolean option that specifies that the data requested is the name of the Wrapper wrapper used to define the configuration element. The name is constructed assuming the metadata data was declared with no Wrapper reference property to shared declaration. The -reference_name option is used to get the name of the element referenced.

- -meta_object

  An optional Boolean option that specifies that the data requested is a collection containing the Wrapper wrapper used to define the configuration element. The object is the metadata data element as if it was declared with no Wrapper reference property to shared object. The reference_object option is used to get the referenced object.

- -meta_id

  An optional Boolean option that specifies that the data requested is a unique integer corresponding to the Wrapper wrapper used to define the configuration element. The meta_id of an element declared with the Wrapper reference property is the meta_id of the referenced element.

- -partition_name

  An optional Boolean option that specifies that the data requested is the name of the partition in which the configuration element exists. Table 7-10 shows the returned value based on which partition the configuration element exists in.

- -default_value

  An optional option value pair that specifies that the data requested is the default value of a property. The command returns a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the default_value is returned as a Tcl string which is the case for most properties you introspect.

  You can use the -value_id option in combination with the -default_value to obtain the default value of a specific value. When requesting the default value of a single value, the return value is always a well-formatted Tcl string.

- -tolower

  An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All upper case letters are converted to lower case.

- -toupper

  An optional Boolean option that can be used when the command is returning a string. It is ignored otherwise. All lower case letters are converted to upper case.

- -file_name

  An optional Boolean option that specifies that the data requested is the file name from which the configuration element was read. It returns null if the configuration element was not read from a file using read_config_data but instead it was created in memory using add_config_element. The file_name value is updated to point to the output file name as soon as it is written out using the write_config_data command.

- -object

  An optional Boolean option that specifies that the data requested is a collection containing the configuration element just like add_config_element returns when adding an element.

- -partition *partition_name*

  An optional option value pair that is used to specify the partition in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications (DftSpecification and DefaultsSpecification/DftSpecification) are located. You can access the Core wrappers comprising the Tessent Core Description (TCD) file syntax in the tcd partition. Using the partition names meta: and meta:tcd allows you to access the metadata configuration elements that define the default and tcd partitions. Table 7-10 summarizes the available partition names.

Table 7-10. Available Partitions for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the Default partition is accessed which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tessent Core Description file syntax is accessed.
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

Example

This example shows usage of the get_config_value command with a wide variety of its options exercised. You can see the syntax of the configuration data at the top and the specific configuration data being introspected just after. At the bottom of the example, you can see the metadata defining the syntax.

```
//  command: set_context dft -rtl
//  command: report_config_syntax SingularWrapper
SingularWrapper {
  WrapperWithRepeatableId(<name*>) {
    WrapperWithRepeatableId(<name*>) {
      // Same as /SingularWrapper/WrapperWithRepeatableId
    }
    time_property    : <time>        ; // default: 3.5ns
    integer_property : <integer>     ; // default: 3
    binary_property  : <binary>      ; // default: 3'b0
    string_property  : <string>, ... ;
    CsvWrapper(<module_name>,<design_id>) {
      <value>,...;
    }
    DataWrapper_with_no_value {
      <port> ; // repeatable
    }
    DataWrapper_value {
      <port> : <period> ; // repeatable
    }
  }
}
//  command: read_config_data ./data/cfg
//  command: report_config_data SingularWrapper

SingularWrapper {
  WrapperWithRepeatableId(a,b c) {
    WrapperWithRepeatableId(abc) {
      time_property : 4.5ns;
      integer_property : 6;
      binary_property : 4'b110;
      string_property : this is a string;
      CsvWrapper(moda,id1) {
        a;
        b, c;
        d, e f, g;
      }
      DataWrapper_with_no_value {
        a string ;
        1;
      }
      DataWrapper_value {
        clka : 4ns;
        clkb : 6.7ms;
        clkc : 4;
      }
    }
  }
```

```
}
> set wrp [get_config_el WrapperWithRepeatableId(abc) -hier]
```

Case 1:

Get the value of a property where the type is time:

```
> puts [get_config_value time_property -in $wrp]
4.5ns
```

Case 2:

Get the time value in milliseconds with units removed:

```
> puts [get_config_value time_property -in $wrp -time_unit ms]
4.5e-06
```

Case 3:

Get the time value in nanoseconds with units removed:

```
> puts [get_config_value time_property -in $wrp -time_unit ns]
4.5
```

Case 4:

Get the value of a property where the type is integer:

```
> puts [get_config_value integer_property -in $wrp]
6
```

Case 5:

Get the value of a property where the type is binary:

```
> puts [get_config_value binary_property -in $wrp]
4'b110
```

Case 6:

Get the value of a property with a binary value converted in hexadecimal:

```
> puts [get_config_value binary_property -in $wrp -number_format hex]
4'h6
```

Case 7:

Get the value of a property with a binary value converted to integer:

```
> puts [get_config_value binary_property -in $wrp -number_format int]
6
```

Case 8:

Get the value of a property where the type is sting:

```
> puts [get_config_value string_property -in $wrp]
this is a string
```

Case 9:

Get the leaf_name of a wrapper:

```
> puts [get_config_value $wrp -leaf_name]
WrapperWithRepeatableId(abc)
```

Case 10:

Get the base_name of a wrapper:

```
> puts [get_config_value $wrp -base_name]
WrapperWithRepeatableId
```

Case 11:

Get a cvs wrapper object and return the number of rows in csv wrapper:

```
> set csv [get_config_value CsvWrapper(moda,id1) -in $wrp -object]
> puts [get_config_element -in $csv -count]
3
```

Case 12:

Get the values of CSV rows:

```
> puts [get_config_value CsvWrapper(moda,id1)/<0> -in $wrp]
a
> puts [get_config_value CsvWrapper(moda,id1)/<1> -in $wrp]
b c
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp]
d {e f} g
```

Case 13:

Get the number of values in a row:

```
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp -value_count]
3
```

Case 14:

Get specific value of a CSV row:

```
> puts [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp -value_id <1>]
e f
```

Above is equivalent to

```
> puts [lindex [get_config_value CsvWrapper(moda,id1)/<2> -in $wrp] 1]
e f
```

Case 15:

Get the number of the id for a wrapper element:

```
> puts [get_config_value -id_count \
          SingularWrapper/WrapperWithRepeatableId<0>]
2
```

Case 16:

Get the leaf_name of a wrapper element:

```
> puts [get_config_value -leaf_name $wrp]
WrapperWithRepeatableId(abc)
```

Case 17:

Get the name of a wrapper element:

```
> puts [get_config_value $wrp]
SingularWrapper/WrapperWithRepeatableId(abc)
```

Case 18:

Get the base_name of a wrapper element:

```
> puts [get_config_value -base_name $wrp]
WrapperWithRepeatableId
```

Case 19:

Get the parent_name of a wrapper element:

```
> puts [get_config_value -parent_name $wrp]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 20:

Get the parent_object of a wrapper element:

```
> puts [get_single_name [get_config_value -parent_object $wrp]]
/SingularWrapper/WrapperWithRepeatableId(a,b c)
```

Case 21:

Get the root_name of a wrapper element:

```
> puts [get_config_value -root_name $wrp]
/SingularWrapper
```

Case 22:

Get the root_object of a wrapper element:

```
> puts [get_single_name [get_config_value -root_object $wrp]]
/SingularWrapper
```

Case 23:

Get the file_name of a wrapper element:

```
> puts [get_config_value -file_name $wrp]
/home/abc/data/ex.cfg
```

Case 24:

Gets the meta_name of a wrapper element:

```
> puts [get_config_value -meta_name $wrp]
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)
```

Case 25:

Gets the ref_name of a metadata element. Returns null when the element is not referencing another one:

```
> puts [get_conf_value \
       SingularWrapper/WrapperWithRepeatableId/WrapperWithRepeatableId \
       -part meta -ref_name]
/Wrapper(SingularWrapper)/Wrapper(WrapperWithRepeatableId)
> puts [get_conf_value SingularWrapper/WrapperWithRepeatableId  \
       -part meta -ref_name]
```

Case 26:

Uses -lolower and -toupper options to convert the returned string in lower case or upper case:

```
> puts [get_config_value -base_name $wrp]
WrapperWithRepeatableId
> puts [get_config_value -base_name $wrp -toupper]
WRAPPERWITHREPEATABLEID
> puts [get_config_value -base_name $wrp -tolower]
wrapperwithrepeatableid
```

Case 27:

Reports the metadata that defines the syntax being introspected:

```
> report_config_data SingularWrapper -partition meta
Wrapper(SingularWrapper) {
  Repeatable : single;
  Wrapper(WrapperWithRepeatableId) {
    id(name*) {
      type : string;
    }
    Reference = /SingularWrapper/WrapperWithRepeatableId;
    Property(time_property) {
      Value(time) {
        type : time;
        DefaultValue : 3.5ns;
      }
    }
    Property(integer_property) {
      Value(integer) {
        type : int;
        DefaultValue : 3;
      }
    }
    Property(binary_property) {
      Value(binary) {
        type : binary;
        DefaultValue : 3'b0;
      }
    }
    Property(string_property) {
      Value(string*) {
        type : string;
        DefaultValue : "";
      }
    }
    CSVWrapper(CsvWrapper) {
      Repeatable : on;
      id(module_name) {
        Type : String;
      }
      id(design_id) {
        Type : String;
        IgnoreCase : On;
      }
    }
    DataWrapper(DataWrapper_with_no_value) {
      RowIdElement(port) {
```

```
            Type : String;
            repeatable : on;
          }
        }
        DataWrapper(DataWrapper_value) {
          RowIdElement(port) {
            Type : String;
            repeatable : on;
          }
          Value(period) {
            Type : time;
          }
        }
      }
    }
```

Related Commands

set_config_value

add_config_element

get_context

Context: unspecified, all contexts

Mode: all modes

Usage

get_context [-scan | -ijtag | -scan_diagnosis | -extraction | -atpg | -edt | -logic_bist | -rtl | -no_rtl] -design_identifier

Description

Introspects the current context.

When specified without an option, this command returns dft or patterns. When specified with one of the available options, this command returns 1 if the specified sub-context is active.

Arguments

- -scan

  An option that returns 1 when set_context -scan is specified.

- -ijtag

  An option that returns 1 when set_context -ijtag is specified.

- -scan_diagnosis

  An option that returns 1 when set_context -scan_diagnosis is specified.

- -extraction

  An option that returns 1 when set_context -ijtag is specified. When set_current_design is executed, the extraction sub-context is set to 1 if the ICL module for the current design has not already been loaded; in this case, it will be extracted when set_system_mode analysis is executed.

- -atpg

  An option that returns 1 when set_context -scan is specified. When set_system_mode analysis is executed and no Logic BIST configuration is detected, the -atpg sub-context is set to 1.

- -edt

  An option that returns 1 when set_context -scan is specified. When set_system_mode analysis is executed and the EDT configuration is detected, the -edt sub-context is set to 1.

- -logic_bist

  An option that returns 1 when set_context -scan is specified. When set_system_mode analysis is executed and a Logic BIST configuration is detected, the -logic_bist sub-context is set to 1.

- -rtl

  An option that returns 1 when set_context -rtl is specified.

- -no_rtl

  An option that returns 1 when set_context -no_rtl is either specified or inferred. It is inferred in all patterns contexts and in the dft -scan and -edt contexts.

- -design_identifier

  An option that returns the specified or inferred design_identifier value.

Return Value

Returns dft or patterns when no option is specified. Returns 0 or 1 when an options is specified.

Example

In the following example, a script introspects the current context and displays an appropriate message.

```
if {[get_context] == "dft"} {
   if {[get_context -scan]} {
      puts "in dft scan context"
   } elseif {[get_context -edt]} {
      puts "in dft edt context"
   } else {
      if {[get_context -rtl]} {
         puts "in dft rtl context"
      } else {
         puts "in dft no_rtl context"
      }
   }
} else { ;# patterns -> get_context is patterns
   if {[get_context -scan]} {
      if {[get_context -atpg]} {
         if {[get_context -edt]} {
            puts "in patterns scan edt atpg context"
         } else {
            puts "in patterns scan atpg context"
         }
      } else {;# logic_bist -> get_context -logic_bist is 1
         puts "in patterns scan logic_bist context"
      }
   } elseif {[get_context -ijtag]} {
      if {[get_context -extraction]} {
         puts "in ijtag icl_extraction context"
      } else {
         puts "in ijtag retargeting context"
      }
   } else {;# scan_diagnostic
      puts "in scan_diagnostic context"
   }
}
```

Related Commands

report_context

set_context

get_current_design

Context: all contexts

Mode: all modes

Usage

get_current_design [-icl]

Description

Returns a collection of one element that specifies the top-level design when previously set.

The get_current_design command returns an empty collection when the top-level design has not been previously defined using the set_current_design.

Arguments

- -icl

  An optional switch that returns the name of an ICL module object if that object exists. The switch returns an error if there is no ICL module associated with the current design.

Example

```
get_current_design
{ }

read_verilog top.v
set_current_design top
get_current_design
{top}

puts [get_current_design]
@45

puts [get_single_name [get_current_design]]
top

if { [get_single_name [get_current_design]]== "top" } { puts "yes" }
yes
```

The first get_current_design command returns an empty collection because no current design has been set. The second get_current_design command returns a collection consisting of the name of the top-level design that was previously set. The first “puts” command returns only the pointer (@45) to a collection in memory. The second “puts” command shows how to use the get_single_name command to return the name of the module object returned by get_current_design.

Related Commands

set_current_design

get_name_list

get_current_mode

Context: patterns -scan, patterns -scan_retargeting, patterns -scan_diagnosis, patterns -failure_mapping

Mode: setup, analysis

Usage

get_current_mode [-type]

Description

Returns the current mode as specified by the set_current_mode command.

If an argument is not specified with this command, the command returns the name of the mode. If the -type argument is specified, this command returns the mode type.

Arguments

- -type

  An optional switch that specifies to return the mode type. Possible return values are: unwrapped, internal, external, and retargeting. The retargeting return value applies only to the patterns -scan_retargeting and -failure_mapping contexts. For more information about these modes, see the set_current_mode command.

Example

This example returns the mode as “unwrapped”.

get_current_mode

```
unwrapped
```

Related Commands

set_current_mode

get_current_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

get_current_simulation_context

Description

Returns the name of the current simulation context.

The get_current_simulation_context command returns the current simulation context. For a list of predefined simulation contexts, refer to Table 7-40 on page 1648.

Arguments

None.

Example

The following example shows how to set and return the current simulation context:

```
SETUP> get_ current_simulation_context
stable_after_setup

SETUP> set_current_simulation_context stable_load_unload
SETUP> get_ current_simulation_context
stable_load_unload
```

Related Commands

add_simulation_context
copy_simulation_context
delete_simulation_contexts
get_simulation_context_list
report_simulation_contexts
set_current_simulation_context

get_defaults_value

Context: unspecified, dft, patterns

Mode: setup, analysis, insertion

Usage

get_defaults_value *property_path_name*
[-policy *policy*]
[-time_unit *time_unit*] [-number_format *format*]
[-is_specified] [-tolower] [-toupper]

Description

Obtains the default value of a property as specified in one of the DefaultsSpecification/DftSpecification(company | group | user) wrappers.

When the -policy option is not specified, it returns the value specified in the DefaultsSpecification/DftSpecification's user policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

Arguments

- *property_path_name*

  A required string that defines the path to a property without the "DefaultsSpecification()/" prefix.

- -policy *policy*

  An optional option value pair that defines from which DefaultsSpecification wrapper the default value is requested. If not specified, the command returns the value specified in the user policy, the value specified in the group policy if not specified in the user policy, the value specified in the company policy if not specified in the user or group policy, or the tool default if not specified in any policy.

- -time_unit *time_unit*

  An optional option value pair that is only relevant when obtaining the value of a property of type time. This option allows you to request the time expressed in the units you want. For more information see the description of the get_config_value -time_unit option.

- -number_format *format*

  An optional option value pair that is only relevant when obtaining the value of a property of type binary. This option allows you to request the binary number expressed in the format you want. For more information see the description of the get_config_value -number_format option.

- -is_specified

  An optional Boolean option that specifies that you are not requesting the actual value of the property but instead requesting status on whether the property is specified or not in the

given policy. If the -policy option is not specified, the tool returns 1 if it is specified in any of the policy wrappers. If the -policy option is not specified and the tool returns 0, then you know you are getting the tool default value.

- -tolower

  An optional option value pair that is only relevant when obtaining the value of a property of type string. This option allows you to request the string with all upper-case letters converted to lower case.

- -toupper

  An optional option value pair that is only relevant when obtaining the value of a property of type string. This option allows you to request the string with all lower-case letters converted to upper case.

Example

The following example queries the default value of a property and then checks which policy specified it:

```
> set p DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in
> get_defaults_value $p
my_scan_in
> foreach policy {user group company} {
  if {[get_defaults_value $p -policy $policy -is_specified]} {
    break
  }
}
>puts "'$p' is specified by policy $policy'."
'DftSpecification/IjtagNetwork/HostScanInterface/Block/scan_in' is
specified by policy 'group'.
```

Related Commands

set_defaults_value

create_dft_specification

get_design_level

Context: dft

Mode: setup, analysis, insertion

Usage

get_design_level

Description

Returns the design level previously defined with the set_design_level command.

When a new design is elaborated, the level goes back to being unspecified. In this case, the get_design_level command returns a null string.

Arguments

None

Example 1

The following example queries the level of the current design and stores the result in a variable.

```
set design_level [get_design_level]
```

Related Commands

set_design_level

set_current_design

get_design_sources

Context: unspecified, all contexts

Mode: setup

Usage

get_design_sources [-format { verilog | icl }] [-path_list | -extension_list]

Description

Returns the pathnames or file extensions previously specified with the set_design_sources command.

By default, the get_design_sources command returns a list of paired values that specify the pathnames to all design sources. The first item in the pair is -v or -y to indicate whether the pathname is a file or a directory, respectively. This is useful when you want to add additional design sources because any invocation of the set_design_sources command overwrites previous invocations.

Arguments

- -format { verilog | icl }

  An optional switch that allows you to choose Verilog modules or ICL modules (Tessent Shell only). By default, the command specifies Verilog modules.

- -path_list | -extension_list

  An optional switch that specifies to return either a list of pathnames or a list of file extensions. By default, the command returns a list of pathnames.

Example 1

This example returns the list of pathnames and then the list of extensions:

```
set_design_sources -y Path1 Path2 -ext EXT1 -y Path3 -ext EXT2 EXT3
get_design_sources -path_list
-y Path1 -y Path2 -y Path3

get_design_sources -extension_list
EXT1 EXT2 EXT3
```

Example 2

This example shows the difference between the set_design_sources and get_design_sources output, and demonstrates how to incrementally add to the list of design sources:

SETUP> **set_design_sources**

```
//  -------------------------------------------------------
//  List of verilog -v/-y files/directories in search order
//  -------------------------------------------------------
//  MODB.v
//  MODC.v
//  MODD.v
//  -------------------------------------------------------
```

SETUP> **get_design_sources**

```
-v MODB.v -v MODC.v -v MODD.v
```

SETUP> **set_design_sources -v MODB.v -v MODC.v**
SETUP> **get_design_sources**

```
-v MODB.v -v MODC.v
```

SETUP> **set_design_sources [get_design_sources] -v MODD.v**
SETUP> **get_design_sources**

```
-v MODB.v -v MODC.v -v MODD.v
```

Related Commands

report_design_sources

set_design_sources

get_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Usage

get_dfm_rules [-header | -col *columns*...]

Description

Returns a TCL list of the DFM rules contained in the layout database.

By default, for each rule the tool returns the id, rule name, layer, and type.

For more information about DFM rules, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- -header

  An optional switch that returns a list of available column names.

- -col *columns*...

  An optional switch that specifies the column names you want to use for the DFM rule information in the list.

Example1

The following example sets the available column names of the DFM report table into a TCL variable and displays the column names.

```
open_layout MyLayoutDB
set columns [ get_dfm_rules -header ]
puts "columns: $columns"
```

```
columns: id rule layer type
```

Example 2

The following example creates a custom report by inspecting the DFM rules:

```
foreach { id rule_name layer_name type_name }  [ get_dfm_rules ] {
    puts "$id $rule_name $layer_name $type_name"
}
```

```
1 minimum_space_4_line_route1 route_1 BRIDGE
2 metal_cross_edge_route1 route_1 OPEN
3 low_density_min_space_route2 route_2 BRIDGE
4 half_comb_route3 route_2 OPEN
5 metal_cross_edge_route1 route_3 OPEN
6 corner_to_interior_metal4 route_4 BRIDGE
7 divergent_line_with_jog_metal5 route_5 BRIDGE
8 metal_cross_edge_route1 route_5 OPEN
9 route_5_barbell route_5 OPEN
10 route_2_between_vias route_2 OPEN
```

Example 3

The following example customizes the columns for the report:

```
foreach { r t }  [ get_dfm_rules -col rule type ] {
    puts "$r $t"
}
```

```
minimum_space_4_line_route1  BRIDGE
metal_cross_edge_route1  OPEN
low_density_min_space_route2  BRIDGE
half_comb_route3  OPEN
metal_cross_edge_route1  OPEN
corner_to_interior_metal4  BRIDGE
divergent_line_with_jog_metal5  BRIDGE
metal_cross_edge_route1  OPEN
route_5_barbell  OPEN
route_2_between_vias  OPEN
```

Related Commands

report_dfm_rules

get_dft_cell

Context: dft

Mode: all modes

Usage

get_dft_cell -available_cell_selection_names | *function_name*
[-cell_selection_name *dft_cell_selection_name*] [-inputs number] [-silent]

Description

Retrieves the named sets of cells to use when test hardware must be inserted (as a Tcl list of the currently available cell selection names), or a particular cell given the name and desired cell function of a specified set of cells.

If only one set (one cell_selection) exists, then the name of the cell selection is optional for retrieving a cell of a given function. Otherwise, -cell_selection_name is used to identify a specific cell selection.

This command is very useful in enabling you to write a technology-independent command. After you have retrieved the cell module using the get_dft_cell command, you can use the function attribute on the ports of the module to find the ports you want to connect to.

For more information on Tessent cell library attributes, see the *Tessent Cell Library Manual*. More specifically, refer to the dft_cell_selection wrapper description described in the Cell Selection section as well as the function attribute described in Defining Pin Attributes.

You can see a list of the available DFT cell functions by using the “help get_dft_cell” command. Each of the listed functions corresponds one for one with the elements documented in the dft_cell_selection wrapper found in the Cell Selection section of the *Tessent Cell Library Manual*. Many basic functions are learned and automatically assigned to an unnamed dft_cell_selection wrapper when your library does not contain a dft_cell_selection wrapper.

Note

If your Tessent cell library contains only ATPG model information but you have a cell.lib file in the original LogicVision format, you can load it natively using the read_cell_library command and the cell_selection section will be populated.

Arguments

- -available_cell_selection_names

  An optional switch that specifies for the tool to return a Tcl list of names of the currently available DFT cell selections. If none are parsed and one is learned by default, it will return a list with the single element “default”. If there is no cell selection parsed and no default is created (for example, if no library is loaded) it will return null.

- *function_name*

  A required string that specifies the DFT cell function from the specified or default dft_cell_selection to return. If the specified function does not exist in the specified dft_cell_selection, the tool issues an error and lists the dft_cell_selection names that contains the requested function_name.

- -cell_selection_name *dft_cell_selection_name*

  A switch and string pair that specifies the name of a currently available cell selection to retrieve cells from. This could be a technology name, for example. This switch is optional if only one dft_cell_selection wrapper is currently loaded. If this switch is not specified and more than one dft_cell_selection wrapper exists, the tool issues an error. Similarly, if an invalid technology is specified, the tool issues an error and lists all available dft_cell_selection names.

- -inputs *number*

  An optional switch and integer pair that specifies the number of inputs on the model. The default number is 2. The default is ignored except in the case of symmetrical primitives {and, nand, or, nor, xor}. The default for symmetrical primitives is 2 unless the -inputs number pair specifies an integer greater than 2; in this case the tool returns a cell with the specified number of inputs if one exists in the specified technology.

- -silent

  An optional switch that specifies to suppress the error message and returns an empty collection.

Return Value

Returns the module objects when no errors are present.

Example 1

In the following example, the command queries which cell model is available in the tech_name_1 dft_cell_selection wrapper to be used as a clock_buffer.

get_dft_cell clock_buffer -cell_selection_name tech_name_1

```
{my_buff}
```

Example 2

In the following example, the command queries which 4-input cell model is available to be used as an and gate.

get_dft_cell and -input 4

```
{and_4}
```

Example 3

In the following example, multiple dft_cell_selection wrappers exist but the -cell_selection_name switch is not specified. As a result, the tool returns an error and lists the available dft_cell_selection names that exist in the library.

```
SETUP> get_dft_cell and
//  Error: Multiple cell_selection(<selection_name>) exist.
//         Must specify -cell_selection_name followed by one of :
//         tech_name_1
//         tech_name_2
//         tech_name_3
```

Example 4

In the following example, an invalid technology is specified. As a result, the tool returns an error and lists the available technologies.

```
get_dft_cell and -cell_selection_name foo
//  Error: cell_selection(foo) does not exist.
//         -cell_selection_name must currently be followed by one of :
//         tech_name_1
//         tech_name_2
//         tech_name_3
```

Example 5

In the following example, a procedure called intercept_with_cell_body is created to allow intercepting ports, nets, or pins that are either sinks or drivers. The second argument is a cell_selection type string that uses the get_dft_cell command to obtain the corresponding cell from the library, thereby making this command technology-independent. To understand the purpose of each section, read the comments in the commented section headers.

Note, in Section 3, the port attribute called ‘function’ is used to determine the port names to connect on the selected dft cell. You can also refer to the example for the register_tcl_command to see how you can create a cleaner external interface to this command with proper option parsing and help messages.

```
#######################################################################
### This array is to not repeat interception into repeated module
### instances when auto_uniquify_edited_modules is off
#######################################################################

variable insertion_parent_leaf_instance_array
array set insertion_parent_leaf_instance_array {}
proc intercept_with_cell_body {args} {

   # Intitialize the allArgs array with the default values of the optional
   # options
   array set allArgs {
      input2 ""
      select ""
      level 1
      clear 0
   }

   #Override the default values of the arguments with the actual values
   # received from
   array set allArgs $args
   set node $allArgs(node)
   set cell_sel $allArgs(cell_sel)
   set in2 $allArgs(input2)
   set sel $allArgs(select)
   set level $allArgs(level)
   set clear $allArgs(clear)

   #######################################################################
   ### Section 1: Defines a global array to know if an interception was
   ### already done inside repeated module instance. clear args set to 1
   ### resets the array
   #######################################################################

   variable insertion_parent_leaf_instance_array

   if {$clear == 1} {
      array unset insertion_parent_leaf_instance_array
      array set insertion_parent_leaf_instance_array {}
   }
   set dft_cell [get_dft_cell $cell_sel]
```

```
#####################################################################
### Section 2: Validates node argument and determine if it is a sink or
### driver interception
#####################################################################

set node_object [check_obj_spec $node <node>]
set node_type   [get_attribute_value_list $node_object \
                  -name object_type]
if {$node_type eq "port" } {
   if {[get_attribute_value_list $node_object -name direction] \
         eq "input"} {
      set insertion_type "driver"
   } else {
      set insertion_type "sink"
   }
   set leaf_name [lindex  [get_attribute_value $node_object \
         -name name] 0]
   set instance_object ""
   set leaf_block_id ""
} elseif {$node_type eq "pin"} {
   if {[get_attribute_value_list $node_object -name direction] \
         eq "output"} {
      set insertion_type "driver"
   } else {
      set insertion_type "sink"
   }
   set instance_object [get_instances -of_pins $node_object]
   set leaf_name [lindex [get_attribute_value $node_object \
         -name leaf_name] 0]
   set leaf_block_id [lindex [get_attribute_value $instance_object \
         -name leaf_block_id] 0]
} else {
   set insertion_type "driver"
   set instance_object [get_instances -of_nets $node_object]
   set leaf_name [lindex [get_attribute_value $node_object \
         -name leaf_name] 0]
   set leaf_block_id [lindex [get_attribute_value $node_object \
         -name leaf_block_id] 0]
}

#####################################################################
### Section 3: Determine if in2 and sel are needed
#####################################################################

set connect_in2 0
set connect_sel 0
if {$cell_sel == "mux" | $cell_sel == "clock_mux"} {
   set input0 "[get_single_name [get_port -of_module $dft_cell \
            -filter {function == mux_in0}]]"
   set input1 "[get_single_name [get_port -of_module $dft_cell \
            -filter {function == mux_in1}]]"
   set select "[get_single_name [get_port -of_module $dft_cell \
            -filter {function == mux_select}]]"
   set output "[get_single_name [get_port -of_module $dft_cell \
            -filter {function == data_out}]]"
   set connect_in2 1
   set connect_sel 1
} elseif {$cell_sel == "buffer" | $cell_sel == "clock_buffer" | \
```

```
            $cell_sel == "inverter" } {
      set input0 "[ get_single_name [get_port -of_module $dft_cell \
            -filter {function == data_in}]]"
      set output "[ get_single_name [get_port -of_module $dft_cell \
            -filter {function == data_out}]]"
} else {
   set input0 "[lindex [get_name_list [get_port -of_module $dft_cell \
      -filter {function == data_in}]] 0]"
   set input1 "[lindex [get_name_list [get_port -of_module $dft_cell \
      -filter {function == data_in}]] 1]"
   set output "[get_single_name [get_port -of_module $dft_cell \
      -filter {function == data_out}]]"
   set connect_in2 1
}

if {$connect_in2} {
   if {$in2 != ""} {
      set in2_object [check_obj_spec $in2 "-input2"]
   } else {
      return -code error "-input2 must be specified when \
         -cell_function_name is two input cell"
   }
}

if {$connect_sel} {
   if {$sel != ""} {
      set sel_object [check_obj_spec $sel "-select"]
   } else {
      return -code error "-select must be specified when \
         -cell_function_name is mux or clock_mux"
   }
}

#####################################################################
### Section 4: Determines location of the interception gate and if it
### was already done for the specified node inside its parent module.
#####################################################################

if {[sizeof_collection $instance_object] ne 0} {
   if {$node_type eq "net"} {
      set leaf_instance_name ""
      set parent_module [get_modules -of_instances $instance_object]
      set parent_instance_name [get_single_name $instance_object]
   } else { # it is a pin
      set leaf_instance_name [lindex [get_attribute_value_list \
         $instance_object -name leaf_name_hash] 0]
      set parent_instance [get_instances -parent_of_instances \
         $instance_object]
      set parent_instance_name [get_single_name $parent_instance]
      if {[sizeof_collection $parent_instance] eq 0} {
         set parent_module [get_current_design]
      } else {
         set parent_module [get_modules -of_instances \
            $parent_instance]
      }
   }
} else {
   set parent_instance_name ""
```

```
      set leaf_instance_name ""
      set parent_module [get_current_design]
   }
   set parent_mod_name [get_single_name $parent_module]
   set parent_lib_name [lindex [get_attribute_value_list $parent_module \
      -name library_name] 0]
   set parent_arch_name [lindex [get_attribute_value_list $parent_module \
      -name vhdl_architecture] 0]

   ######################################################################
   # This key will match for insertion made inside repeated instances of
   # a module or inside a generate loop block. When it match, the
   # interception is skipped for the next invokation
   ######################################################################

   set key "Level${level}-Cell(${cell_sel})-"
   append key \
      "${parent_lib_name}::${parent_mod_name}(${parent_arch_name})"
   if {$leaf_instance_name ne ""} {
      append key "/${leaf_instance_name}"
   }
   append key "/${leaf_name}"

   #######################################################################
   ### Section 5: Retrieves leaf instance name of gate if node is a
   ### repeated module instance and skips the interception
   ######################################################################

   if {[info exists insertion_parent_leaf_instance_array($key)] && \
         [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
      set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
      if {$leaf_block_id ne ""} {
         set cell_inst [get_instance \
            "${parent_instance_name}/${leaf_block_id}.${leaf_cell_inst}"]
      } else {
         set cell_inst [get_instance \
            "${parent_instance_name}/${leaf_cell_inst}"]
      }
   } else {
      if {$leaf_block_id ne ""} {
         set cell_inst_name \
         "${parent_instance_name}/${leaf_block_id}.insertion_${cell_sel}"
      } else {
         set cell_inst_name \
            "${parent_instance_name}/insertion_${cell_sel}"
      }
      no_transcript {
         set cell_inst [create_instance $cell_inst_name -of_module \
            $dft_cell -allow_instance_uniquification]
      }

      set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
         leaf_name] 0]
      set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
         -name leaf_block_id] 0]
      if {$leaf_block_id ne ""} {
         # Remove the block_id part from the leaf instance name
         set leaf_name [string range $leaf_name \
```

```
            [string length $leaf_block_id] end]
         }
         set insertion_parent_leaf_instance_array($key) $leaf_name

         if {$insertion_type == "sink"} {
            no_transcript {
               move_connections -from $node_object \
                  -to [get_pins ${input0} -of_instances  $cell_inst]
               create_connections [get_pins ${output} -of_instances \
                  $cell_inst] $node_object
            }
         } else {
            no_transcript {
               move_connections -from $node_object -to [get_pins ${output} \
                  -of_instances $cell_inst]
               create_connections $node_object [get_pins ${input0} \
                  -of_instances  $cell_inst]
            }
         }
      }

      if {$connect_in2} {
         no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
               $cell_inst]
         }
      }

      if {$connect_sel} {
         no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
               $cell_inst]
         }
      }
      return $cell_inst
   }

   ##########################################################################
   # This procs checks that the value is either an single element collection
   # of nets, ports, or pins or the name of a single net, port, or pin.
   ##########################################################################

   proc check_obj_spec {obj_or_name id} {
      if {[is_collection $obj_or_name]} {
         if {[sizeof_collection $obj_or_name] ne 1} {
            display_message "Supplied collection for ${id} has \
               more than one element."
            return -code error
         } else {
            set obj_type [lindex [get_attribute_value_list $obj_or_name \
               -name object_type] 0]
            set object $obj_or_name
            if {$obj_type ne "port" && $obj_type ne "pin" && \
               $obj_type ne "net"} {
               display_message "Specified value for ${id} has \
                  type '$obj_type' instead of 'pin', 'port' or 'net'"
               return -code error
            }
```

```
         }
      }else {
         set net_object [get_nets  $obj_or_name -silent]
         if {[sizeof_collection $net_object] eq 0} {
            display_message "Specified value for ${id} is not an \
               existing 'pin', 'port' or 'net' name."
            return -code error
         } elseif {[sizeof_collection $net_object] ne 1} {
            display_message "Specified value for ${id} is more \
               than one bit wide"
            return -code error
         }
         set port_object [get_ports $obj_or_name -silent]
         set pin_object [get_pins  $obj_or_name -silent]
         if {[sizeof_collection $port_object] ne 0} {
            #If the name is also a port, pick the port object
            set object $port_object
         } elseif {[sizeof_collection $pin_object] ne 0} {
            #If the name is also a pin, pick the pin object
            set object $pin_object
         } else {
            #The net is not a port nor a pin
            set object $net_object
         }
      }
      return $object
   }
```

Related Commands

report_context

set_context

get_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Usage

get_fanins ***obj_spec*** [-stop_on cell_pin | net | pin] [-silent]

Description

Returns a collection of all requested objects found in the fanin of the specified pin, net, or port object.

When obj_spec is an output or inout pin of a cell, then the command reports all input or inout pins on the same cell that are in the combinational fanin of the output or inout pin. Note that the fanin direction of an inout pin is toward the inside of the instance, and the fanout direction of an inout pin is toward the outside of the instance.

The -stop_on option specifies object types upon which the tracing should stop. It does not mean that the returned collections contain only objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net will be reported as the object on the fanin of the pin even if -stop_on is specified as cell_pin. If the net had not been dangling and it had a driver, the get_fanins command would not have stopped on the net and would have continued to walk the fanin path.

The get_fanins command returns an empty collection when there is no fanin. The fanin of an input port is null.

The get_fanins command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the trace_flat_model command to perform tracing on the flat model with simulation conditions considered.

Arguments

- ***obj_spec***

  A required value that specifies a net, pin, port, or pseudo_port name; a Tcl list of net, pin, port, or pseudo_port names; or a collection of net, pin, port, or pseudo_port objects.

- -stop_on cell_pin | net | pin

  An optional switch and literal pair that specifies the following:

  - When -stop_on is set to cell_pin (the default), the fanin walk traverses nets and design pins but stops on pins of cells or primitives in the fanin of the object. The walk may still, however, stop on a net or a design pin when the net or design pin does not have a fanin.
  - When -stop_on is set to net, the fanin walk traverses through one level of “assign” statements and stops on the net connected to the start pin.

- When -stop_on is set to pin, the fanin walk traverses nets and reports the pin in the fanin of the object. This could be an input or inout pin above the object or an output or inout pin on child instances. The walk may still, however, stop on a net if the net does not fanin to pins.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Example 1

The following example shows a series of get_fanins command invocations of the circuit example shown in Figure 7-13.

```
# find net connected to pin u4/I5
get_fanins u4/I5 -stop_on net

{n3}

#Find cell_pin in the fanin of u4/I5
get_fanins u4/I5

{u3/Y}

#Find pin in the fanin of u4/I5. Returns cell pins as there is no design pins before it
get_fanins u4/I5 -stop_on pin

{u3/Y}

#Find pin in the fanin of u4/u1/A0
get_fanins u4/u1/A0 -stop_on pin

{u4/I1}

#Find all nets connected to the D pin of dff modules
get_fanins [get_pins -filter {module_name == “dff” && leaf_name == “D”}] -stop_on net

{I3 n6 u4/n2 u4/n3 u4/u5/n1 u4/u5/I3 u5/n1}

#Find pins in the fanin of an output pin of a cell
get_fanins u1/Y

{u1/A0 u1/A1 u1/A2}
```

Example 2

The following example adds an internal primary input (pseudo_port). Based on the example circuit shown in Figure 7-13, it then returns the collection of objects found in the fanin of both of the pseudo_port’s underlying pins.

```
add_primary_inputs {u4/I4 u4/I5} -internal -pin_name user_pi1
get_fanins user_pi1

{I5 u3/Y}
```

Figure 7-13. Circuit Illustrating the Usage of the get_fanins Command

Related Commands

get_fanouts
get_icl_ports
get_icl_pins
get_nets

get_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Usage

get_fanouts ***obj_spec*** [-stop_on cell_pin | net | pin] [-silent]

Description

Returns a collection of all requested objects found in the fanout of the specified pin, net, or port object.

When obj_spec is an input pin of a cell, then the command reports all output or inout pins on the same cell that are in the combinational fanout of the input pin. Note that the fanout direction of an inout pin is toward the outside of the instance, and the fanin direction of an inout pin is toward the inside of the instance.

The -stop_on option is used to specify object types upon which the tracing should stop. Specifying this option does not mean that the returned collections will only contain objects of the specified type, because the tracing may stop before reaching the specified type. For example, a pin may be connected to a dangling net. The net will be reported as the object on the fanout of the pin even if -stop_on is specified as cell_pin. If the net had been connected to other pins, the get_fanouts command would not have stopped on the net and would have continued to walk the fanout path.

The get_fanouts command returns an empty collection when there is no fanout.The fanout of an output or inout port is null.

The get_fanouts command does not use simulated values and only considers connectivity of the hierarchical data model. You can use the trace_flat_model command to perform tracing on the flat model with simulation conditions considered.

Arguments

- ***obj_spec***

  A required value that specifies a net, pin, port, or pseudo_port name; a Tcl list of net, pin, port, or pseudo_port names; or a collection of net, pin, port, or pseudo_port objects.

- -stop_on cell_pin | net | pin

  An optional switch and literal pair that specifies the following:

  - When -stop_on is set to cell_pin (the default), the fanout walk traverses nets and design pins but stops on pins of cells or primitives in the fanout of the object. The walk may still, however, stop on a net or a design pin when the net or design pin does not have a fanout.

  - When -stop_on is set to net, the fanout walk traverses through one level of “assign” statements and stops on the net connected to the start pin,.

- When -stop_on is set to pin, the fanout walk traverses nets and reports the pin in the fanout of the object. This could be output or inout pins above the object or inputs or inout pins on child instances. The walk may still, however, stop on a net if the net does not fanout to pins.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Example 1

The following example shows a series of get_fanouts command invocations of the circuit example shown in Figure 7-14.

#Find cell pins in the fanout of u4/I5
get_fanouts u4/I5

```
{u4/u4/A2 u4/u5/u3/D}
```

#Find pins in the fanout of u4/u1/Y. Returns a cell pin as there is no design pin before it
get_fanouts u4/u1/Y -stop_on pin

```
{u4/u2/D}
```

#Find pins in the fanout of u4/u4/Y
get_fanouts u4/u4/Y -stop_on pin

```
{u4/u1/A2 u4/u5/I2}
```

#Find all nets connected to the Q pin of dff modules
get_fanouts [get_pins -filter {module_name == “dff” && leaf_name == “Q”}] -stop_on net

```
{n2 O3 u4/n3 u4/O1 u4/u5/O1 u4/u5/O2 u5/n2}
```

#Find net in fanout of a cell output pin
get_fanouts u1/Y -stop_on net

```
{n1}
```

Example 2

The following example adds an internal primary output (pseudo_port). Based on the example circuit shown in Figure 7-14, it then returns the collection of objects found in the fanout of the pseudo_port’s underlying pin.

add_primary_outputs {u4/I5} -internal
get_fanouts [get_port u4/I5 -of_type pseudo]

```
{u4/u4/A2 u4/u5/u3/D}
```

Figure 7-14. Circuit Illustrating the Usage of the get_fanouts Command

Related Commands

get_fanins
get_icl_ports
get_icl_pins
get_nets

get_fault_type

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

Mode: all modes

Usage

get_fault_type

Description

Returns the current fault model type, as previously specified by the set_fault_type command.

The command returns one of the following strings: stuck, toggle, iddq, transition, path_delay, bridge, udfm.

Arguments

- None

Example 1

The following example shows how to set the fault model type and then introspect the fault model:

ANALYSIS> **set_fault_type bridge**
ANALYSIS> **get_fault_type**
bridge

Example 2

The following example loads an SDC file if the current fault model type is “transition”:

```
if {[get_fault_type] eq "transition" } {
   read_sdc myDesign.sdc
} else {
   puts "No SDC file read. Fault type is [get_fault_type]."
}
```

Related Commands

set_fault_type

get_gate_pins

Context: all contexts

Mode: setup, analysis

Prerequisites: Although you can use this command in all system modes, you can only use it with a flattened netlist. In Tessent Shell, netlist flattening typically happens when you first attempt to enter analysis mode or when you issue the create_flat_model command.

Usage

get_gate_pins [*gate_pin_id_list* | *hier_pin_name_patterns*] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent] [-of_instances *instance_objects* | -below_instances *instance_objects* | -of_pins *pin_objects* | -of_ports *port_objects*]

Description

Returns a collection of gate_pin objects found below the current design in the flat model.

The command supports arbitrary filtering based on attribute values. It also supports those options for frequently used filters.

Arguments

- *gate_pin_id_list* | *hier_pin_name_patterns*

  An optional string that specifies a list of gate_pin objects. The gate_pin_id_list is a list of gate_pin IDs (two integers separated by a period, the first integer representing a gate and the second representing a gate_pin). No wild cards are allowed in the gate_pin_id_list. If the second integer is not present, the ID specifies all pins on the gate.

  The hier_pin_name_patterns is a list of strings (wildcard patterns or regular expressions) matching hierarchical pin names. Using the hier_pin_name_patterns is equivalent to filtering on the hier_pin_name attribute.

Note

When the gate_get_pin command operates on the flat model, it removes all backslashes and white spaces from escaped identifiers. You must not include backslashes or white spaces in the *hier_pin_name_patterns* string. If you want to access the corresponding gate_pin, use the gate_pin_id attribute of the hierarchical pin object. If you want to access the gate_pin by name, use the get_name_list -remove_escaping command and switch.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the gate_pin object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operators used simple wild card matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the hier_pin_name_patterns argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for gate_pins matching the value of the hier_pin_name_patterns argument.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns.You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified instance_objects.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the circuitry located underneath the specified instance_objects.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified pin_objects.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return only gate_pins associated with the specified port_objects or pseudo_port objects.

Example 1

The following example filters the collection of all gate_pin objects based on an attribute_equation of user-defined attributes:

```
get_gate_pins -filter {myType1 || myType2 == "ABC"}
{/u1/U2/b 16.0 16.1 16.2 24.0}
```

Example 2

The following example returns the collection of all gate_pin objects matching a list of case-insensitive hierarchical pin name patterns. For more examples of pattern matching with regular expressions, refer to "Glob and Regular Expression Pattern Matching Syntax" on page 107.

```
get_gate_pins {u1/u1/a u*/b} -nocase
{/u1/U1/a /u1/U1/b /u1/U2/b /u1/U3/b /u2/U1/b}
```

Example 3

The following example returns a collection of all gate_pin objects on the gate with ID=12, the output gate_pin object (pin_index=0) on the gate with ID=14, and the gate_pin object with pin_index=2 on the gate with ID=11:

get_gate_pins { 12 14.0 11.2 }
{/u2/U1/b 12.0 14.0 /u1/U3/c}

Example 4

The following equivalent get_gate_pins commands return a collection of all gate_pin objects where the corresponding hierarchical pin name matches the wildcard pattern /u1/*/a or /u2/U1*:

get_gate_pins { /u1/*/a /u2/U1* }
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}

get_gate_pins -filter "hier_pin_name =~ /u1/*/a || hier_pin_name =~ /u2/U1*"
{/u1/U1/a /u1/U2/a /u1/U3/a /u2/U1/b /u2/U1/c /u2/U1/a /u2/U1/y /u2/U1/z}

Example 5

The following example adds to the fault list all of the gate_pin objects corresponding to the hierarchical pins matching the wildcard pattern u1/*:

add_faults [get_gate_pins u1/*]
report_faults

```
type    code    pin_pathname
----   ------   ------------------------------------
  0     UC      /u2/U1/b
  1     UC      /u2/U1/b
  0     UC      /u2/U1/c
  1     UC      /u2/U1/c
  0     UC      /u2/U1/a
  1     UC      /u2/U1/a
  0     UC      /u2/U1/y
  1     UC      /u2/U1/y
  0     UC      /u2/U1/z
  1     UC      /u2/U1/z
```

Example 6

The following examples are based on the circuit in Figure 7-13 on page 891. This example returns all of the gate pins for the specified instance u4/u2:

SETUP> **get_gate_pins -of_instances u4/u2**

```
{u4/u2/Q u4/u2/QB u4/u2/CLK u4/u2/D}
```

This example returns all of the gate pins named D that are found below the u4 level of hierarchy:

SETUP> **get_gate_pins */D -below_instances u4**

```
{u4/u2/D u4/u3/D u4/u5/u2/D u4/u5/u3/D}
```

Related Commands

get_fanins
get_common_parent_instance
get_modules
get_nets

get_icl_extraction_options

Context: all contexts

Mode: all modes

Usage

get_icl_extraction_options -extract_learned_muxes | -extract_library_muxes

Description

Provides access to the settings specified by the set_icl_extraction_options command.

Arguments

- -extract_learned_muxes

  A switch that instructs the tool to return the extract_learned_muxes setting of the ICL extraction options. Possible values are 0 and 1.

- -extract_library_muxes

  A switch that instructs the tool to return the extract_library_muxes setting of the ICL extraction options. Possible values are 0 and 1.

Example

The following command returns the setting specified by the –extract_library_muxes option of set_icl_extraction_options command:

get_icl_extraction_options –extract_library_muxes

```
1
```

Related Commands

set_icl_extraction_options
report_icl_extraction_options

get_icl_fanins

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Usage

get_icl_fanins ***obj_spec*** [-stop_on none | pin] [-silent]

Description

Returns a collection of all requested objects found in the fanin of the specified pin or port objects.

The -stop_on option is used to specify object types that the tracing should stop on. Specifying this option does not mean that the returned collections will only contain objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be directly connected to a port. The port will be reported as the object on the fanin of the pin even if -stop_on is specified as pin.

Arguments

- ***obj_spec***

  A required value that specifies an ICL pin or port name; a Tcl list of ICL pin or port names; or a collection of ICL pin or port objects.

- -stop_on none | pin

  An optional switch and literal pair that specifies the following:

  none — The fanin walk traverses the connections and the pins in the fanin of the object. The trace reports pins with no fanin or ports reached in the fanin walk.

  pin — The fanin walk traverses the connection and reports the pin in the fanin of the object; this could be an input pin above the object or an output pin on child instances. The walk may still, however, stop on a port if the pin does not fanin to pins.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Example

The following example shows command invocations with and without the -stop_on pin option to illustrate its effect.

```
# find true fanin of pin block1.sib1.s1
get_icl_fanins block1.sib1.s1

{si} # Went thought block1.si as -stop_on was defaulted to none. Reached
ICL port si on the current_design
```

```
# find first pin in fanin of block1.sib1.s1
```
get_icl_fanins block1.sib1.s1 -stop_on pin

```
{block1.si} # Stopped on block1.si as -stop_on was specified as pin.
```

Related Commands

get_fanouts
get_icl_ports
get_icl_pins

get_icl_fanouts

Context: all contexts

Mode: all modes

Prerequisites: The current design must be set.

Usage

get_icl_fanouts ***obj_spec*** [-stop_on none | pin] [-silent]

Description

Returns a collection of all requested objects found in the fanout of the specified ICL pin, or port objects.

The -stop_on option is used to specify object types that the tracing should stop on. Specifying this option does not mean that the returned collections will only contain objects of the specified type because the tracing may stop before reaching the specified type. For example, a pin may be directly connected to a port. The port will be reported as the object on the fanout of the pin even if -stop_on is specified as pin.

Arguments

- ***obj_spec***

  A required value that specifies an ICL pin or port name; a Tcl list of ICL pin or port names; or a collection of ICL pin or port objects.

- -stop_on none | pin

  An optional switch and literal pair that specifies the following:

  none — The fanout walk traverses the connections and the pins in the fanout of the object. The trace will report pins with no fanout or ports that were reached in the fanout walk.

  pin — The fanout walk traverses the connection and reports the pin in the fanout of the object. This could be an output pin above the object or an input pin on child instances. The walk may still, however, stop on a port if the pin does not fanout to pins.

- -silent

  An optional switch that suppresses the error that is normally generated if any object within obj_spec does not exist, in which case it is ignored.

Example

The following example shows the get_icl_fanouts command invocations with and without the -stop_on pin option to illustrate its effect.

```
# find true fanout of pin block1.sib1.so
get_icl_fanins block1.sib1.so
```

```
{so} # Went thought block1.so as -stop_on was defaulted to none. Reached
ICL port so on the current_design

# find first pin in fanout of block1.sib1.so
```
get_icl_fanins block1.sib1.so -stop_on pin

```
{block1.so} # Stopped on block1.so as -stop_on was specified as pin.
```

Related Commands

get_fanins
get_icl_ports
get_icl_pins

get_icl_instances

Context: all contexts

Mode: all modes

Usage

Usage 1: Without the -hierarchical option

get_icl_instances [*name_patterns*] [-of_modules *obj_spec*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *icl_pin_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Usage 2: With the -hierarchical option

get_icl_instances [*leaf_name_patterns*] [-of_modules *obj_spec*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects*]
[-hierarchical] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Usage 3: With no name pattern

get_icl_instances [-of_modules *obj_spec*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *icl_pin_objects*]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ICL instances instantiated relative to the current design that match the specified *name_patterns list or -filter expression.

Usage 1: The name patterns are expressed as hierarchical instance path names relative to the current design. You can have any number of wildcards or regular expressions in the names between the hierarchy separator symbols as shown in the examples for this command. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

Usage 2: The name patterns are leaf instance names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf instance name pattern is searched relative to all ICL instances found below the current design.

Usage 3: A name_patterns argument is not specified so that all ICL instances are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all ICL instances that match the results returned by the -filter attribute_equation filter as shown in Usage 3. In Usage 2, described previously, the name patterns cannot include slashes or dots which are hierarchy separator symbols. In Usage 1, you can use the dot or slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

- -of_modules *obj_spec*

  An optional switch and value pair that constrains the command to return only instances of ICL modules specified by *obj_spec*. The *obj_spec* is a single ICL module name, a Tcl list of ICL module names, or a collection of ICL module objects as returned by the get_icl_modules command.

- -parent_of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return a collection of ICL instance objects that are the parent instances of the specified instance_objects. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional name_patterns option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -parent_of_instances option is used.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for ICL instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -of_pins *icl_pin_objects*

  An optional switch and value pair that constrains the command to return the ICL instance to which the ICL pins belong. The returned collection is filtered to return any given ICL instance only once, even if the instance belongs to many ICL pins in the icl_pin_objects list. When this option is specified, the optional name_patterns must be leaf_name patterns that are checked against the leaf name of the ICL instances. An error is generated if the specified name_patterns contains more than one level of hierarchy when the -of_pins option is used.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the name_patterns argument cannot include slashes or dot which are hierarchy separator symbols.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL instance object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on the filtering attribute equation format.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. You must properly escape the bracket "[]" symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax. You must use the slash as a hierarchy separator when using the -regexp option.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

The following example illustrates Usage 3 and returns the collection of all instances of ICL modules modb and modd found anywhere below the current design:

get_icl_instances -of_modules {modb modd}

```
{u2 u1/u5}
```

Example 2

The following example illustrates Usage 3 in which the collection of all instances is filtered based on an attribute_equation of user-defined attributes.

get_icl_instances -filter {myType1 || myType2 == "ABC"}

```
{u2 u1.u5}
```

Example 3

The following example illustrates Usage 1 in which the collection of all instances matching a list of hierarchical name patterns is returned. The names u1/* and u*/u* are only searched relative to the current design.

get_icl_instances {u1/* u*/u*}

```
{u4.u5 u1.u7}
```

Example 4

The following example illustrates Usage 2 in which the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

get_icl_instances u3* -hierarchical

```
{u3 u4.u30 u4.u5.u31 u5.u3}
```

Related Commands

get_icl_modules
get_icl_pins
get_icl_ports
read_icl

get_icl_modules

Context: all contexts

Mode: all modes

Usage

get_icl_modules [*name_patterns*] [-of_instances *instance_objects* |
-below_instances *instance_objects*] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ICL modules that match the specified name_patterns list or -filter expression.

By default, this command returns all ICL modules below the current design. You can use the -of_instances argument to constrain the search.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ICL modules. If no name_patterns are specified, the tool searches all ICL modules that match the results returned by the -filter attribute_equation filter.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL modules of the specified *instance_objects*. The instance_objects is a single ICL instance name, a Tcl list of ICL instance names, or a collection of ICL instance objects as returned by the get_icl_instances command. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL modules corresponding to instances found below the specified *instance_objects*.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL module object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the name_patterns.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

The following example returns ICL modules whose name matches a* or b*.

get_icl_modules {a* b*}

```
{ab abc b bcd}
```

Example 2

The following example returns ICL modules having one or two characters.

get_icl_modules {[:alnum:]{1,2}} -regexp

```
{ab b}
```

Related Commands

get_icl_pins
get_icl_instances
get_icl_pins
get_icl_ports
read_icl

get_icl_module_parameter_list

Context: all contexts

Mode: all modes

Usage

get_icl_module_parameter_list ***module_object*** [-overridden]

Description

Returns the list of parameters in an ICL module.

Arguments

- ***module_object***

  A required string that specifies the parameterized ICL module objects whose parameters are to be returned.

- -overridden

  An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Value

List of parameters.

Example

n the following example, this command is used to obtain the list of parameters overridden in the specific instantiation. Notice how the get_icl_instances command is used to find the ICL instance u1/myinst_i1; this ensures that the get_attribute_value_list command returns the module_name attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]
set params [get_icl_module_parameter_list [get_attribute_value_list $icl_inst\
                -name module_name] -overridden]
```

Related Commands

get_icl_module_parameter_value

get_icl_module_parameter_value

Context: all contexts

Mode: all modes

Usage

get_icl_module_parameter_value ***parameter_name*** [-of_module *icl_module_object*]

Description

Returns the value of the parameter on the specified module and indicates whether it is the default value or is overridden.

Arguments

- ***parameter_name***

  A required string that specifies the name of the parameter whose value is to be returned.

- -of_module *icl_module_object*

  An optional switch and string pair that specifies the ICL module whose parameter value is to be returned. When omitted, the paramters of the current design are introspeced.

Return Value

The value of the requested ICL module parameter.

Example

In the following example, the values of all overridden parameters are echoed in the transcript. Notice how the get_icl_instances command is used to find the ICL instance u1/myinst_i1; this ensures that the get_attribute_value_list command returns the module_name attribute of the ICL instance and not the design instance in case a design instance of the same name exists.

```
set icl_inst [get_icl_instance u1/myinst_i1]
set icl_mod [get_attribute_value_list $icl_inst -name module_name]
set params [get_icl_module_parameter_list $icl_mod -overridden]
foreach param $params {
        puts “parameter $param = [get_icl_module_parameter_value $param \
            -of_mod $icl_mod]”
}
```

Related Commands

get_icl_module_parameter_list

get_icl_pins

Context: all contexts

Mode: all modes

Usage

Usage 1: Without the -hierarchical option

get_icl_pins *name_patterns* [-direction input | output]
[-function *function*]
[-of_instances *instance_objects* | -below_instances *instance_objects*]
[-filter *attribute_equation*]
[-regexp] [-nocase] [-silent]

Usage 2: With the -hierarchical option

get_icl_pins *leaf_name_patterns* | *leaf_instance/leaf_name_patterns* -hierarchical
[-below_instances *instance_objects*] [-filter *attribute_equation*]
[-direction input | output] [-function *function*]
[-regexp] [-nocase] [-silent]

Usage 3: With no name pattern

get_icl_pins [-filter *attribute_equation*] [-direction input | output] [-function *function*] [-regexp]
[-nocase] [-silent]

Description

Returns a collection of all hierarchical ICL pins instantiated relative to the current design that match the specified name_patterns list or -filter expression.

Usage 1: The name patterns are expressed as hierarchical pin pathnames relative to the current design. You can have any number of wild cards or regular expressions in the names between the hierarchy separator symbols as shown in the examples that follow. You can use the dot or the slash as the hierarchy separator when not using the -regexp option. You can only use the slash as the hierarchy separator when using the -regexp option because the dot is the wildcard symbol when using regular expressions.

Usage 2: The name patterns are leaf pin names or leaf instance name/leaf pin names with any number of wildcards or regular expressions. Unlike in Usage 1, the leaf name pattern is searched relative to all instances found below the current design. There can only be one hierarchy separator when using the -hierarchical option as shown in Example 3.

Usage 3: A name_patterns argument is not specified so all ICL pins are selected and then filtered by the other criteria specified by the other optional arguments.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns | leaf_instance/leaf_name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all ICL pins that match the results returned by the -filter attribute_equation filter as shown in Usage 3. In Usage 2, described previously, the name patterns are not allowed to include more than one slash or dot which are hierarchy separator symbols. See section "Attribute Filtering Equation Syntax" on page 105 for a complete description of the pattern syntax when the -regexp option is used or not.

- -direction input | output

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only ICL pins of the specified instance_objects. The instance_objects is a single ICL instance name, a Tcl list of ICL instance names, or a collection of ICL instance objects as returned by the get_icl_instances command. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for ICL pins below the specified *instance_objects.* Specified name_patterns are searched relative to the specified *instance_objects* when this option is used.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all ICL instances found below the current design. When this option is specified, the name_patterns argument cannot include more then one slash or dot which are the hierarchy separator symbols.

- -function *function*

  An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL pin object. When omitted, pins with any function are considered. The value *function* is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL pin object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operator use

simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ICL pin names matching *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

The following example illustrates Usage 3 where the collection of all ICL pins is filtered based on an attribute_equation of user-defined attributes.

get_icl_pins -filter {myType1 || myType2 == "ABC"}

```
{u1.Y u4.u4.A0 u7.D}
```

Example 2

The following example illustrates Usage 1 where the collection of all ICL pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for examples of pattern matching with regular expressions.

get_icl_pins {u1/Y u*/Q}

```
{u1.Y u2.Q u7.Q}
```

Example 3

The following example illustrates Usage 2 where the collection of all ICL pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all ICL instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

get_icl_pins u*/Y* -hierarchical

```
{u1.Y u3.Y u4.u1.Y u4.u4.Y u4.u5.u1.Y u5.u1.Y u5.u3.Y u5.u4.Y u6.Y}
```

Example 4

The following example gets all ICL pins with function clock or to_clock in the ICL data base.

get_icl_pins -function {clock to_clock}

```
{PLL1.ref PLL1.vco mbist.clk}
```

Related Commands

get_common_parent_instance
get_icl_ports
get_iproc_argument_default
read_icl

get_icl_ports

Context: all contexts

Mode: all modes

Usage

get_icl_ports [*name_patterns*] [-of_modules *obj_spec*] [-direction input | output] [-function *function*] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all ports on a given module that match the specified name_patterns list or -filter expression.

You can have any number of wildcards or regular expressions in the name patterns.

If you issue this command but have not previously loaded any icl files, it will return an empty collection.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ports. If no name_patterns are specified, the tool searches all ports that match the results returned by the -filter attribute_equation filter, if one is specified.

- -of_modules *obj_spec*

  An optional switch and value pair that constrains the command to return only ports on modules specified by *obj_spec*. The *obj_spec* is a single ICL module name, a Tcl list of ICL module names, or a collection of one or more module objects as returned by the get_icl_modules command.

- -direction input | output

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- -function *function*

  An optional switch and literal pair that specifies to filter results based on the value of the function attribute on the ICL port object. When omitted, ports with any function are considered. The value *function* is a Tcl list of function names derived from the ICL port function where multiple words are separated by underscores instead of using camel casing. For example, a port declared as a ScanInPort in ICL has function scan_in in the ICL data model.

- -filter *attribute_equation*

  An optional switch and string that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the ICL port object type. See section “Attribute Filtering Equation Syntax” on page 105 for

more details on filtering attribute equation format. The =~ and !~ matching operators used simple wild card matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument and the arguments of the =~ and !~ -filter expressions as regular expressions instead of as simple wildcard patterns. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

This example looks for ICL ports on the current design whose name match I* or O*.

get_icl_ports {I* O*}

```
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

get_icl_ports {[[:alnum:]]{1,2}} -regexp -of_module ModA

```
{ab a b}
```

Related Commands

get_common_parent_instance
get_icl_pins
get_iproc_argument_default
read_icl

get_icl_scan_interface_list

Context: dft, patterns -ijtag

Mode: setup, analysis

Usage

get_icl_scan_interface_list

Description

Returns the names of the scan interfaces in an existing ICL top module, if the ICL top module exists. If the ICL top module does not exist, it returns the names of the scan interfaces that will be created in the new ICL top module by ICL extraction after the add_icl_scan_interfaces command has been used to specify the scan interfaces.

You must set the current design with set_current_design before using the get_icl_scan_interface_list command.

Arguments

- None

Return Value

If the ICL top module exists, this command returns the ScanInterface names in the ICL top module. If the module does not have an explicit ScanInterface definition in the ICL module, but only an implicitly defined anonymous scan interface, the command returns “unnamed”. If the ICL top module does not exist, the command returns the new ScanInterface names specified with the add_icl_scan_interfaces command, which will be created during ICL extraction.

Example

The following example returns the ScanInterface names PG1 and PG2 in the ICL module pass1:

```
Module pass1 {
   ScanInPort sin1;
   ScanInPort sin2;
   ScanOutPort sout;
   ShiftEnPort sen;
   CaptureEnPort cen;
   SelectPort select1;
   SelectPort select2;
   UpdateEnPort uen;
   ScanInterface PG1 {
        Port sin1; Port sout; Port sen; Port cen; Port select1; Port uen;
   }
   ScanInterface PG2 {
        Port sin2; Port sout; Port sen; Port cen; Port select2; Port uen;
   }
}
```

set_current_design pass1

get_icl_scan_interface_list

```
PG1 PG2
```

Related Commands

add_icl_scan_interfaces
delete_icl_scan_interfaces
get_icl_scan_interface_port_list
set_icl_scan_interface_ports

get_icl_scan_interface_port_list

Context: dft, patterns -ijtag

Mode: setup, analysis

Usage

get_icl_scan_interface_port_list **-name** ***name***

Description

Returns the names of the ports for the specified scan interface in an existing ICL top module, or the names of the ports for the specified scan interface that will be created in the new ICL top module during ICL extraction.

You must set the current design with set_current_design before using the get_icl_scan_interface_port_list command.

Arguments

- **-name** ***name***

  A required switch and value pair that specifies the name of scan interface for which the list of ports will be returned. If the module does not have an explicit ScanInterface definition in the ICL module, then it has an implicitly defined anonymous scan interface. In this case, use the string “unnamed” for the scan interface name to get the ports of the anonymous scan interface.

Return Value

If the ICL top module exists, this command returns the names of the ports for the specified ScanInterface.

If the ICL top module does not exist, the command returns the names of the new ports for the ScanInterface specified with the set_icl_scan_interface_ports command, which will be created during ICL extraction.

Example 1

This example returns the list of ports for the ScanInterface named default in the ICL module pass1:

```
Module pass1{
   ScanInPort sin;
   ScanOutPort sout;
   SelectPort select;
   CaptureEnPort cen;
   ShiftEnPort shiftEn;
   UpdateEnPort updateEn;
   ScanInterface default {
      Port sin; Port sout; Port select; Port cen; Port shiftEn; Port
updateEn; DefaultLoadValue 3'b0;
   }
```

set_current_design pass1

get_icl_scan_interface_port_list -name default

```
sin sout select cen shiftEn updateEn
```

Example 2

This example returns the list of ports for the anonymous scan interface in the ICL module pass2:

```
Module pass2{
   ScanInPort sin;
   ScanOutPort sout;
   SelectPort select;
   CaptureEnPort cen;
   ShiftEnPort shiftEn;
   UpdateEnPort UpdateEn;
   }
```

set_current_design pass2

get_icl_scan_interface_port_list -name unnamed

```
sin sout select cen shiftEn updateEn
```

Related Commands

add_icl_scan_interfaces
delete_icl_scan_interfaces
get_icl_scan_interface_list
set_icl_scan_interface_ports

get_icl_scope

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

get_icl_scope [-iCall | -iPrefix]

Description

Introspects the current ICL scope.

When executed without an option, this command returns the instance name prefix that will be appended to the beginning of any ICL object names used in the iCall, iWrite, and iRead commands. When executed with the -iCall option, this command ignores the current iPrefix specification and returns the ICL scope against which the current iProc is running. When executed with the -iPrefix option, this command returns the current value specified for the iPrefix command.

For more information on the ICL scope, see the iCall command reference page.

Arguments

- -iCall

  An optional literal that specifies to ignore the current iPrefix specification and return the ICL scope against which the current iProc is running. This path is useful to obtain the exact ICL instance the iProc was executed against using the iCall command. You can use this value in an iNote as shown in the example below.

- -iPrefix

  An optional literal that specifies to return the current value specified for the iPrefix command.

Return Value

An ICL instance path name.

Example

In the following example, the iProc gets the ICL scope in which it was executed against with the iCall command to echo into the retargeted pattern using an iNote command:

```
iProc myProc {} {
        iNote “Running on Instance [get_icl_scope -iCall]”
}
```

Related Commands

iCall

iPrefix

get_iclock_list

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

get_iclock_list [-overridden]

Description

Returns a list of ICL port and pin names on which you have issued an iClock command.

When you use the -overridden option, this command lists the ICL port and pin names on which you have issued an iClockOverride command on. You can use this list to introspect the effective source and the effective frequency multiplication and division values of the ICL pin or port using the get_iclock_option [-overridden] command.

Arguments

- -overridden

  Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

Return Value

A Tcl list of ICL port and pin names.

Example

In the following example, a Tcl list of ICL port and pin names are obtained using the get_iclock_list command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
        puts “iClock pin or port: $iclock”
        puts “ source = [get_iclock_option $iclock -source]”
        puts “ freq_multiplier= [get_iclock_option $iclock -freq_multiplier]”
        puts “ freq_divider = [get_iclock_option $iclock -freq_divider]”
}
```

Related Commands

add_clocks
delete_clocks
get_iclock_option
iClock
iClockOverride
iRunLoop
report_clocks
report_icl_modules

get_iclock_option

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

get_iclock_option ***icl_port_or_pin_name*** [-overridden]
{**-source** | **-freq_multiplier** | **-freq_divider** | **-period** [-in { fs | ps | ns | us | ms | s }] }

Description

Returns the effective or specified source, frequency multiplier, frequency divider, or clock period values for the specified icl_port_or_pin_name.

The specified icl_port_or_pin_name must have been the target of an iClock command when the -overridden option is not specified; it must have been the target of an iClockOverride command when the -overridden option is specified.

- ***icl_port_or_pin_name***

  A required string that specifies the name of an ICL port or pin object on which an iClock or iClockOverride command has been issued.

- -overridden

  An optional switch that specifies that icl_port_or_pin_name must be an ICL port or pin object on which an iClockOverride command has been issued. When you use this switch, you cannot also use the -period switch.

- **-source**

  A required switch that specifies to return the effective source of an iClock ICL pin or port, or the specified source of an iClockOverride ICL pin or port.

- **-freq_multiplier**

  A required switch that specifies to return the effective frequency multiplier value of an iClock ICL pin or port, or the specified frequency multiplier value of an iClockOverride ICL pin or port.

- **-freq_divider**

  A required switch that specifies to return the effective frequency divider value of an iClock ICL pin or port, or the specified frequency divider value of an iClockOverride ICL pin or port.

- **-period** [-in { fs | ps | ns | us | ms | s }]

  A required switch that returns the clock period of the specified icl_port_or_pin_name. You can optionally use the -in switch and keyword to specify a particular time unit. When you use the -period switch, you cannot also use the -overridden switch. By default, the time unit is derived from the time unit of the source clock. This is either the time unit used in the -period switch of the add_clocks command (in case of an asynchronous source clock) or the time unit of the tester period (in case of a synchronous source clock).

Return Value

The value of the specified option for the specified icl_port_or_pin_name.

Example

In the following example, a Tcl list of ICL port and pin names are obtained using the get_iclock_list command. Their effective source and their effective frequency multiplication and division values are then echoed to the transcript

```
set iclock_list [get_iclock_list]
foreach iclock $iclock_list {
        puts “iClock pin or port: $iclock”
        puts “ source = [get_iclock_option $iclock -source]”
        puts “ freq_multiplier= [get_iclock_option $iclock -freq_multiplier]”
        puts “ freq_divider = [get_iclock_option $iclock -freq_divider]”
}
```

Related Commands

add_clocks
delete_clocks
get_iclock_list
iClock
iClockOverride
iRunLoop
report_clocks
report_icl_modules

get_ijtag_instances

Context: dft, patterns

Mode: all modes

Usage

get_ijtag_instances [*name_spec*] [-below_instances *instance_objects*]
[-filter *attribute_equation*] [-silent]

Description

Returns a collection of instances for which there is a matched ICL module.

You can restrict the search to instances below a given collection of instances and filter the collection based on an arbitary filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if no Ijtag instances are returned.

Arguments

- ***name_spec***

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all Ijtag instances.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering using the attribute equation format.

- -silent

  An optional switch that suppresses the warning message normally generated when the command returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example

The following example uses the get_ijtag_instances command to get all instances of a module below a given instance for which there is a matched ICL file.

```
> get_ijtag_instances -below_instance subblockA_i1
{subblockA_i1/tdr1 subblockA_i1/tdr2}
```

Related Commands

get_ijtag_instance_option

set_ijtag_instance_options

report_ijtag_instances

get_ijtag_instance_option

Context: dft, patterns

Mode: all modes

Usage

get_ijtag_instance_option **ijtag_instance** [-use_in_dft_specification]

Description

Returns the value of the selected option for the specified Ijtag instance.

Arguments

- ***ijtag_instance***

  A required literal that is the name of an ijtag instance or a collection containing one ijtag instance as reported by get_instances or get_ijtag_instances.

  The instance object must have an ICL module matched to it.

- -use_in_dft_specification

  A Boolean option that specifies that the value of the -use_in_dft_specification option is required. This value is set using the set_ijtag_instance_options and is used by the create_dft_specification command.

Example

The following example returns the value of the use_in_dft_specification option of all Ijtag instances.

```
> foreach_in_collection inst [get_ijtag_instances] {
  set opt [get_ijtag_instance_option $inst -use_in_dft_spec]
  puts "inst = [get_single_name $inst], use_in_dft_spec = $opt"
}
inst = u1/trd1, use_in_dft_spec = auto
inst = u1/trd2, use_in_dft_spec = auto
inst = u2/trd1, use_in_dft_spec = off
```

Related Commands

get_ijtag_instances

set_ijtag_instance_options

report_ijtag_instances

get_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Usage

get_ijtag_retargeting_options -test_setup_network_end_state |
-compare_constant_capture_values | -compare_constant_data_out_ports

Description

Returns the value of the specified option of the previous set_ijtag_retargeting_options command or its default value if the set_ijtag_retargeting_options command was not issued.

To use this command, you must have previously loaded and elaborated the top-level ICL module. Otherwise, the tool issues an error message.

Arguments

- -test_setup_network_end_state

  A switch that returns the value of the -test_setup_network_end_state option (keep or reset) specified by the previous set_ijtag_retargeting_options command or returns the default value (reset) if the set_ijtag_retargeting_options command was not issued.

- -compare_constant_capture_values

  A switch that returns the value of the -compare_constant_capture_values option (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (on) if the set_ijtag_retargeting_options command was not issued.

- -compare_constant_data_out_ports

  A switch that returns the value of the -compare_constant_data_out_ports option for the primary DataOut ports (on or off) specified by the previous set_ijtag_retargeting_options command or returns the default value (off) if the set_ijtag_retargeting_options command was not issued.

Example

The following example loads the ICL, turns off the reset of the ICL network after the last test_setup PDL command and then displays the value of the selected option:

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl ../data/chip.icl
SETUP> set_current_design

//  Note: Top design is 'chip'.

SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep
SETUP> get_ijtag_retargeting_options -test_setup_network_end_state

keep
```

Related Commands

report_ijtag_retargeting_options

set_ijtag_retargeting_options

get_insertion_option

Context: unspecified, all contexts

Mode: all modes

Usage

get_insertion_option {-auto_uniquify_edited_modules | -edited_module_prefix | -net_uniquification_suffix | -module_uniquification_suffix | -instance_uniquification_suffix | -allow_assigns | -inserted_object_comment_label}

Description

Returns the value of any insertion option that is specifiable with the set_insertion_options command.

Only one option can be specified per get_insertion_option invocation. The return value is a string containing the current setting of the specified option. An error is generated if more than one option is specified, and the return value will be null.

Arguments

- -auto_uniquify_edited_modules

  A switch that returns the current setting of -auto_uniquify_edited_modules option of the set_insertion_options command.

- -edited_module_prefix

  A switch that returns the current setting of the -edited_module_prefix option of the set_insertion_options command.

- -net_uniquification_suffix

  A switch that returns the current setting of the -net_uniquification_suffix option of the set_insertion_options command.

- -module_uniquification_suffix

  A switch that returns the current setting of the -module_uniquification_suffix option of the set_insertion_options command.

- -instance_uniquification_suffix

  A switch that returns the current setting of the -instance_uniquification_suffix option of the set_insertion_options command.

- -allow_assigns

  A switch that returns the current setting of the -allow_assigns option of the set_insertion_options command.

- -inserted_object_comment_label

  A switch that returns the current setting of the -inserted_object_comment_label option of the set_insertion_options command.

Example

The following example introspects the current setting of the -edited_module_prefix option of the set_insertion_options command.

get_insertion_option -edited_module_prefix

```
MyCore_
```

Related Commands

set_insertion_options

get_insert_test_logic_options

Context: dft -test_points

Mode: setup, analysis

Usage

get_insert_test_logic_options [**-inserted_object_prefix**]

Description

Retrieves the current prefix that will be used when creating test point logic.

The command returns an empy string if no option is specified.

Arguments

- **-inserted_object_prefix**

  A required switch that returns the prefix set with the set_insert_test_logic_options command. If no user-defined prefix has been set, the default prefix “ts_” is returned.

Example

The following example shows how to determine the prefix that will be used when inserting nets and instances during test point insertion. In this case, the command will return "demo1_" because of the previous set_insert_test_logic_options command.

```
set_insert_test_logic_options -inserted_object_prefix demo1_
insert_test_logic
write_design -output netlist.v -replace
set prefix [get_insert_test_logic_options -inserted_object_prefix]
puts "inserted nets and instances will be prefixed with : $prefix"
```

Related Commands

set_insert_test_logic_options

get_instances

Context: all contexts

Mode: all modes

Usage

get_instances [*name_patterns*] [-of_modules *module_objects*]
[-parent_of_instances *instance_objects* | -below_instances *instance_objects* |
-of_pins *pin_objects* | -of_nets *net_objects*]
[-of_type *types*] [-hierarchical] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all instances instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the other optional options.

When the name_patterns list is omitted, all instances are returned subject to filtering imposed by the other optional options.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all instances.

  If you specify one of the optional switches (-parent_of_instances, -of_pins, -of_nets, or -hierarchical), the name patterns are matched against the leaf_name attribute value of the instances in which case no hierarchy separators are allowed in the pattern list.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects otherwise the name patterns are relative to the current design.

- -of_modules *module_objects*

  An optional switch and value pair that constrains the command to return only instances of modules specified by module_objects. The module_objects is a single module name, a Tcl list of module names, or a collection of module objects as returned by the get_modules command.

- -parent_of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return a collection of instance objects that are the parent instances of the specified instance_objects. The returned collection is filtered to only return a given parent instance once. When this option is specified, the optional name_patterns option is a string that is matched against the leaf instance name of every parent instance of the collection. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -parent_of_instances option is used.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the instance on which the pins belong to. The returned collection is filtered to only return a given instance once even if it belongs to many pins in the pin_objects list. When this option is specified, the optional name_patterns is a string that is matched against the leaf instance name of the instances. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -of_pins option is used.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the instance in which the net objects are located. The returned collection is filtered to only return a given instance once even if it is the parent instance of many nets in the net_objects list. When this option is specified, the optional name_patterns is a string that is matched against the leaf instance name of the instances. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -of_nets option is used.

- -of_type *types*

  An optional switch and value pair that constrains the tool to return only instances whose type attribute is one of the following: design, cell, or primitive. The types value is either a single type string or a Tcl list of type strings such as {design cell}.

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all instances found below the current design or below the specified instance_objects when the -below_of_instances option is specified. When this option is specified, the name_patterns argument cannot include slashes which are hierarchy separator symbols.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the instance object type. See section “Attribute Filtering Equation Syntax” on page 105 for more details on filtering attribute equation format.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument as a real regular expression instead of as a simple wildcard pattern. You must properly escape the bracket “[]” symbols so that they are interpreted as part of the regular expression and not interpreted by the Tcl shell. See section “Glob and Regular Expression Pattern Matching Syntax” on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for instances matching the name_patterns.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. However, it is possible the instance targeted by the name_patterns disappeared due to a change in a previous editing step. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

The following example returns a collection of all instances of cell types that start with ABC found anywhere below the current design.

get_instances ABC* -hierarchical -of_type cell

```
{u2/ABCD u1/u5/ABCDEF}
```

Example 2

The following example starts with the collection of all instances and filters it based on an attribute_equation of user-defined attributes.

get_instances -filter {myType1 || myType2 == "ABC"}

```
{u2 u1/u5}
```

Example 3

The following example starts with the collection of all instances and filters it based on the expression specified by the attribute_equation string.

get_instances -filter {name=~"u4*/u1*"}

```
{u4/u1 u4/u5/u1}
```

Example 4

In the following example, the collection of all instances matching a list of hierarchical name patterns are additionally filtered to only return instances of type design. The names u1/* and u*/u* are searched relative to the current design.

get_instances {u1/* u*/u*} -of_type design

```
{u4/u5}
```

Example 5

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* searches for all instances found below the current design.

get_instances u3* -hierarchical

```
{u3 u4/u3 u4/u5/u3 u5/u3}
```

Example 6

In the following example, the collection of all instances matching a leaf name pattern is returned. The leaf instance name u3* is matched against instances found below instance u4.

get_instances u3* -hierarchical -below_instances u4

```
{u4/u3 u4/u5/u3}
```

Example 7

In the following example, the *-parent_of_instances* option is used to find the parent instance of each instance on which a collection of pins exist. Only the parent instances whose leaf instance names match ABC* are returned. As described in the description of the -parent_of_instances switch, the optional name_pattern argument, in this case ABC*, is a string that is matched against the leaf instance name of every parent instance of the collection when the -parent_of_instances option is used.

set pin [get_pins]
get_instances ABC* -parent_of_instances [get_instances -of_pins $pin]

```
{u4/ABC1 u4/u5/ABC2}
```

Alternatively, you could use the base_name attribute of the pin objects and the parent_instance attribute of the instances to achieve the same result. However, the method used in this example is much more efficient because it deals with objects instead of strings containing object names. This method also avoids issues with string concatenation in the presence of escaped identifiers.

Related Commands

create_instance
delete_instances
get_common_parent_instance
rename_instance
replace_instances
uniquify_instances

get_iproc_argument_default

Context: all contexts

Mode: all modes

Usage

get_iproc_argument_default ***proc_name arg_name*** [-exists] [-of_module *module_name*] [-silent]

Description

Returns the default value for the specified arg_name of the specified proc_name attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

This command returns a null string when the specified argument has no default value. If you specify the -exists option, this command returns true if the specified argument has a default value and returns false when it does not.

Arguments

- ***proc_name***

  A required string that specifies the name of the iProc.

- ***arg_name***

  A required string that specifies the name of the argument.

- -exists

  An optional switch that changes the returned value of the command. When present, the command returns true when the specified argument has a default value and returns false when it does not. When not specified, the command returns the default value when it exists; otherwise it returns null.

- -of_module module_name

  An optional switch and value pair that constrains the command to return only the default value for the specified arg_name of the specified proc_name attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified proc_name, arg_name or module_name do not exist.

Return Value

If the -exist option is specified, this command returns true if the specified argument has a default value and returns false when it does not. If the -exist option is not specified, the command returns the default value when it exists; otherwise it returns null.

Example

The following example lists each argument attached to the P1 iProc on module mychip and lists its default value, if one exists; otherwise it reports the argument does not have a default value.

```
foreach arg [get_iproc_argument_list P1 -of_module mychip] {
        if {![get_iproc_argument_default P1 $arg -exist]} {
            puts "Argument $arg has no default value"
        } else {
            set def [get_iproc_argument_default P1 $arg]
            puts "Argument $arg has default value of $def"
        }
}
```

Related Commands

delete_iproc
get_iproc_argument_list
get_iproc_body
get_iproc_list
iProc
iProcsForModule

get_iproc_argument_list

Context: all contexts

Mode: all modes

Usage

get_iproc_argument_list ***iproc_name*** [-of_module *module_name*] [-silent]

Description

Returns the list of argument names for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- ***iproc_name***

  A required string that specifies the name of the iProc for which the arguments are returned.

- -of_module module_name

  An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Value

A Tcl list that contains the argument names of the specified iProc.

Example

The following example assigns the arguments attached to the chiptest iProc on module mychip to the variable chiptest_args:

```
set chiptest_args [get_iproc_argument_list chiptest -of_module mychip]
```

Related Commands

delete_iproc
get_iproc_argument_default
get_iproc_body
get_iproc_list
iProc
iProcsForModule

get_iproc_body

Context: all contexts

Mode: all modes

Usage

get_iproc_body ***iproc_name*** [-of_module *module_name*] [-silent]

Description

Returns the body for the specified iProc attached to the ICL module that was specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- ***iproc_name***

  A required string that specifies the name of the iProc.

- -of_module module_name

  An optional switch and value pair that constrains the command to return only the body of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified iProc or module do not exist.

Return Value

None

Example

The following example returns the body of the chiptest iProc attached to module mychip:

```
get_iproc_body chiptest -of_module mychip
```

Related Commands

delete_iproc
get_iproc_argument_default
get_iproc_argument_list
get_iproc_list
iProc
iProcsForModule

get_iproc_list

Context: all contexts

Mode: all modes

Usage

get_iproc_list [-of_module *module_name*] [-silent]

Description

Returns a Tcl list of iProcs attached to the ICL module specified by the last iProcsForModule command, or to the ICL module specified by module_name.

Arguments

- -of_module *module_name*

  An optional switch and value pair that constrains the command to return only instances of iProcs attached to the module specified by module_name.

- -silent

  An optional switch that specifies to suppress error messages if the specified module does not exist.

Return Value

A Tcl list of the iProc names, which are registered against the ICL module specified by the last iProcsForModule command or the ICL module specified by module_name.

Example

The following example assigns the list of iProcs attached to the mychip ICL module to the variable myprocs:

```
set myprocs [get_iproc_list -of_module mychip]
puts $myprocs
```

Related Commands

delete_iproc
get_iproc_argument_default
get_iproc_argument_list
get_iproc_body
iProc
iProcsForModule

get_license_queue_timeout

Context: all contexts

Mode: all modes

Usage

get_license_queue_timeout

Description

Returns the amount of time Tessent Shell will queue for a required license as specified by the previous set_license_queue_timeout command, or its default value of “unlimited’ if the set_license_queue_timeout command was not issued or it was issued with that value.

The command may also return “no_queue” if that was specified by the set_license_queue_timeout command.

Arguments

- None

Example

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3
SETUP> get_license_queue_timeout

3
```

Related Commands

set_license_queue_timeout

get_logical_library_list

Context: all contexts

Mode: all modes

Prerequisites: You must be in the -rtl context.

Usage

get_logical_library_list

Description

Returns a Tcl list with all logical design library names defined with the set_logical_design_libraries command.

This command and the concept of logical libraries is only available in the RTL mode.

This command is useful for writing scripts that operate on all libraries such as in Example3 of the get_modules command reference.

Arguments

- None

Return Value

A Tcl list with all logical design library names defined with the set_logical_library_list command.

Example

In the following example, the Tcl variable libs is defined by the result returned by the get_logical_library_list command. You can then use a foreach command to iterate on the list elements.

```
set libs [get_logical_library_list]
```

Related Commands

set_logical_design_libraries

get_module_parameter_list

Context: all contexts

Mode: all modes

Usage

get_module_parameter_list ***module_object*** [-overridden]

Description

Returns the list of parameters in a module.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the set_context command.

Arguments

- ***module_object***

  A required string that specifies the parameterized module objects whose parameters are to be returned.

- -overridden

  An optional switch that specifies to only list those parameters whose default values are overridden during an instantiation.

Return Value

List of parameters.

Example

In the following example, the get_module_parameter_list is used to obtain the list of parameters overridden in the specific instantiation.

```
set inst u1/myinst_i1
set params [get_module_parameter_list [get_attribute_value_list $inst\
                -name module_name] -overridden]
```

Related Commands

get_module_parameter_value

get_module_parameter_value

Context: all contexts

Mode: all modes

Usage

get_module_parameter_value ***parameter_name*** [-of_module *module_object*]

Description

Returns the value of the parameter on the specified module object.

This command is only available in -rtl mode (set_context dft -rtl). For more information, see the set_context command.

Arguments

- ***parameter_name***

  A required string that specifies the name of the parameter whose value is to be returned.

- -of_module *module_object*

  An optional switch and string pair that specifies the design module whose parameter value is to be returned. When omitted, the parameters of the current design are introspected.

Return Value

The value of the requested design module parameter.

Example

In the following example, the values of all overridden parameters are echoed in the transcript.

```
set inst [get_instance u1/myinst_i1]
set mod [get_attribute_value_list $inst -name module_name]
set params [get_module_parameter_list $mod -overridden]
foreach param $params {
        puts “parameter $param = [get_module_parameter_value $param \
            -of_mod $mod]”
}
```

Related Commands

get_module_parameter_list

get_modules

Context: all contexts

Mode: all modes

Usage

get_modules [*name_patterns*] [-of_type *type*] [-in_library *library*] [-vhdl_architecture *architecture*] [-of_instances *instance_objects*] [-below_instances *instance_objects*] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all modules that match the specified name_patterns list.

By default, this command returns all modules below the current design. You can use the -of_instances and -below_instances arguments to constrain the search. If no modules exist in memory, the command returns an empty collection.

You can have any number of wildcards or regular expressions in the name patterns.

In the rtl context, you may have more than one logical library. Because VHDL is also supported in the rtl context, you may also have multiple architectures. You define the list of available logical libraries using the set_logical_design_libraries command.

In the no_rtl context, only one library exists and is it named “work”.

The tool searches for the objects specified by the name_patterns argument in all libraries unless the -in_library option is specified. All other commands that have an option that refers to module names only search the default library. For example, you use the following command to create an instance of module M1 from library L1:

```
create_instance instance_name -of_module [get_module M1 -in_library L1]
```

In another example, an instance of module M1 from the default library is created because only the get_modules command knows how to look into other libraries.

```
create_instance instance_name -of_module M1
```

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to match the returned list of modules against. If no name_patterns are specified, the tool searches all modules.

- -of_type *type*

  An optional switch and value pair that constrains the command to return only modules of the specified types. The type is a single string or a Tcl list of stings matching one of the following values: design or cell.

- -in_library *library*

  An optional switch and value pair that specifies to restrict the search to a single library. The library value must be a single string corresponding to one of the logical library names defined with the set_logical_design_libraries command. When the -in_library option is not specified, all libraries are searched. This option is only allowed in the rtl context.

  Using -in_library XXX is equivalent to using -filter “library_name == XXX”

- -vhdl_architecture *architecture*

  An optional switch and value pair that specifies to restrict the search to a specific VHDL architecture name. This option is only useful when searching VHDL modules. When this option is not specified, one module per available architecture is returned. This option is only allowed in the rtl context.

  Using -vhdl_architecture XXX is equivalent to using -filter “vhdl_architecture == XXX”.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only modules of the specified *instance_objects*. You could use the module_name attribute of the instance objects to perform a similar filtering but this method is more direct and efficient and automatically deals with libraries and architectures.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to return only modules corresponding to instances found below the specified *instance_objects*.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the module object type. See section “Attribute Filtering Equation Syntax” on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the name_patterns argument as a real regular expression instead of as a simple wildcard pattern. See section “Glob and Regular Expression Pattern Matching Syntax” on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for modules matching the name_patterns.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a

probable mistake when specifying patterns. However, it is possible the module targeted by the name_patterns disappeared due to a change in a previous editing step. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

The following example looks for modules whose name matches a* or b*.

get_modules {a* b*}

```
{ab abc b bcd}
```

Example 2

The following example looks for modules having one or two characters.

get_modules {[:alnum:]{1,2}} -regexp

```
{ab b}
```

Example 3

The following example gets the module object associated to an instance.

get_modules -of_instances u3/u5

```
{myModule}
```

Example 4

The following example looks for modules in all libraries and displays their names and language value in a tabular format.

```
set format_string "   "
append format_string "module_name = %-10s "
append format_string "language = %-10s "

set libs [get_logical_library_list]

foreach lib $libs {
   puts "Modules in library $lib are:"
   set mods [get_modules -in_library $lib]
   foreach_in_collection mod $mods {
   puts [format $format_string \
      [get_att_val $mod -name name]\
      [get_att_val $mod -name language]]
   }
}

// Command output:

Modules in library lib1 are:
   module_name = dut        language = verilog
Modules in library lib2 are:
   module_name = dut        language = verilog
Modules in library work are:
   module_name = top        language = verilog
```

Related Commands

create_module

get_multiprocessing_option

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Usage

get_multiprocessing_option { **-generic_delete | -generic_scheduler | -license_timeout | -lsf_heuristics | -lsf_learning | -lsf_options | -multithreading | -processors_per_grid_request | -remote_shell | -result_time_limit | -scheduler_timeout | -sge_options** }

Description

Returns the value of a single specified variable previously set with the set_multiprocessing_options command.

The get_multiprocessing_option command returns the value of a single multiprocessing variable that affects the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the set_multiprocessing_options command. To display the values of multiprocessing variables in a human-readable format, use the report_multiprocessing_options command.

For more information about setting up for multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "Using the Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

The arguments for the get_multiprocessing_option command are identical to those of the set_multiprocessing_options command, described on page 1840. For more information about the purpose of each switch, refer to the list of arguments there.

Example

The following example sets and then returns the value of the remote_shell variable:

```
> set_multiprocessing_options -remote_shell rsh
> puts [get_multiprocessing_option -remote_shell]

rsh
```

Related Commands

add_processors
delete_processors
report_multiprocessing_options
report_processors
set_multiprocessing_options

get_name_list

Context: unspecified, all contexts

Mode: all modes

Usage

get_name_list ***object_spec*** [-remove_escaping] [-silent]

Description

Returns the name(s) of the specified object(s).

This command returns a list with the value of the name attribute for the objects specified by *object_spec*. Using this command is equivalent to executing the following command:

```
get_attribute_value_list object_spec -name name
```

The -remove_escaping argument removes the \ and the trailing spaces in an escaped identifier. Many legacy commands such as add_display_instance require the instance name to have the escape character stripped from its name. However, these legacy commands do accept collections as inputs in cases where the name format is not important. See examples below for an illustration of this behavior

If an object in *object_spec* does not exist, an error message displays unless the -silent switch is specified.

Arguments

- ***object_spec***

  A required string that specifies an object name, a Tcl list of object names, or a collection of objects.

- -remove_escaping

  An optional switch that specifies to strip out the \ and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See Example 2 for an illustration of this usage.

- -silent

  An optional switch that specifies to suppress error messages if an object in *object_spec* does not exist.

Example 1

The following example retrieves the names for a collection of pins.

get_name_list [get_pins u1/i*]

```
{u1/i1 u1/i2 u1/i3}
```

Example 2

The following example illustrates the use of the -remove_escaping option. In this example, you can see how legacy commands natively support collections as input arguments but do not support names with escaped identifiers.

Notice the use of the double set of braces when specifying a name pattern containing escape identifiers to get_pins. These braces are needed to prevent the Tcl shell from interpreting the \ character as its own escape sequence. You could also escape the \ using {u2/\\u1/u2 /p*} but using two sets of braces is easier when the pattern includes many slashes. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a complete description of pattern syntax and suggestions to use when the pattern is targeted for names containing escape characters.

get_pins {{u2/\u1/u2 /p*}}

```
{{u2/\u1/u2 /p1} {u2/\u1/u2 /p2}}
```

add_display_instance [get_pins {{u2/\u1/u2 /p*}}] -display design
set names [get_name_list [get_pins {{u2/\u1/u2 /p*}}]]

```
{{u2/\u1/u2 /p1} {u2/\u1/u2 /p2}}
```

add_display_instance $names -display design

```
// Error: Specified instance does not exist or is not available at this
// level of hierarchy.
// Error: Incorrect argument /u2/\u1/u2 /p1.
// Error: Specified instance does not exist or is not available at this
// level of hierarchy.
// Error: Incorrect argument /u2/\u1/u2 /p2.
```

set names [get_name_list [get_pins {{u2/\u1/u2 /p*}}] -remove_escaping]

```
{{u2/u1/u2/p1} {u2/u1/u2/p2}}
```

add_display_instance $names -remove_escaping -display design

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
get_common_parent_instance
get_icl_pins
get_modules
get_icl_ports

get_nets

Context: all contexts

Mode: all modes

Usage

get_nets [*name_patterns*] [-below_instances *instance_objects*]
[-of_pins *pin_objects* |-of_ports *port_objects*]
[-hierarchical] [-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all hierarchical nets instantiated relative to the current design that match the specified name_patterns list, subject to filtering imposed by the other optional options.

When the name_patterns list is omitted, all nets are returned subject to filtering imposed by the other optional options.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of nets. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all nets. In Usage 2 described previously, the name patterns are not allowed to include slashes which are hierarchy separator symbols. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a complete description of the pattern syntax when the -regexp option is used and not used.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for nets below the specified instance_objects. Specified name_patterns are searched relative to the specified instance_objects when this option is used.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the *pin objects*. Those are the nets having the exact same name as the pins and residing inside the instances on which the pins are on. For pins of cells, the inside nets corresponding to the pins are not visible and thus not returned. You cannot use a name_patterns list when using this option.To get the net connected to pins from the outside, use get_fanin and get_fanout -stop_on net.

- -of_ports *port_objects*

  An optional switch and value pair that constrains the command to return the net objects associated to the port_objects. Those are the nets having the exact same name as the ports and reside inside the current design. You cannot use a name_patterns list when using this option.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the net object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operators used simple wild card matching versus Posix extended regular expression based on the presence of the -regexp option or not.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for nets matching the *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. However, it is possible the net targeted by the name_patterns disappeared due to a change in a previous editing step. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

In the following example, the collection of all nets is filtered based on an attribute_equation of user-defined attributes.

get_nets -filter {MyType1 || myType2 == "AB"}

```
{u3_Y u4_O2}
```

Example 2

In the following example, the collection of all nets matching a list of hierarchical name patterns are returned. The names u1/* and u*/u* are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for an example of pattern matching with regular expressions.

get_nets {u1/* u*/u*}

```
{u4/u4_Y u4/u1_Y u4/u2_Q u5/u4_Y u5/u1_Y u5/u2_Q}
```

Example 3

In the following example, the collection of all nets matching a leaf name pattern is returned. The leaf net name u4* searches for all nets found below the current design.

get_nets u4* -hierarchical

```
{u4_O1 u4_O2 u4_O3 u4/u4_Y u5/u4_Y}
```

Related Commands

create_net
delete_pins
delete_nets
get_icl_pins
get_icl_ports

get_open_pattern_set

Context: patterns -ijtag

Mode: analysis

Usage

get_open_pattern_set

Description

Returns the name of the currently open pattern set.

This command returns an empty string if no pattern set is open.

Arguments

- None

Return Value

The name of the currently open pattern set.

Example

The following example prints the name of the currently open pattern set:

```
puts "The curently opened pattern set is called [get_open_pattern_set]"
```

Related Commands

close_pattern_set
get_pattern_set_list
open_pattern_set
report_pattern_sets
reset_open_pattern_set

get_pattern_cycle_count

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

Mode: analysis

Usage

get_pattern_cycle_count

Description

Reports the total number of cycles in the currently loaded WGL or STIL test pattern set.

The get_pattern_cycle_count command reports the total number of cycles in the currently loaded test pattern set. The test pattern set must be STIL or WGL format.

Example 1

The following example sets the pattern source to an external source, the *pat.wgl* file and displays the total number of cycles in the *pat.wgl* test pattern file:

```
read_patterns pat.wgl -wgl
get_pattern_cycle_count
```

Example 2

The following example sets the pattern source to an external pattern database source, the *pat1.wgl* file, appends the pat2.wgl test pattern file to it, and displays the total number of cycles contained in both test pattern files:

```
read_patterns pat1.wgl
read_patterns pat2.wgl –append
get_pattern_cycle_count   //reports total cycles for pat1 and pat2
```

Related Commands

read_patterns

get_pattern_set_data

Context: patterns -ijtag

Mode: analysis

Usage

get_pattern_set_data **pattern_set_name** [-unrolled] { -port_list | -event_list | -event_data *event_id* }

Description

Returns requested details of the internal representation of the specified pattern set.

The **get_pattern_set_data** command provides introspection into the internal representation of retargeted IJTAG patterns. Among other uses, this information can be used to generate pattern files for non-IJTAG compliant access mechanisms (for example, IEEE 1149.7), as long as the access hardware can produce the retargeted IJTAG patterns; that is, as long as the access mechanism does not put any restrictions on IJTAG retargeting.

Arguments

- pattern_set_name

  A required string that specifies the pattern set from which to introspect the requested information.

- -unrolled

  An optional switch that specifies to restrict the information returned when the -event_list or -event_data switches are used. The -unrolled switch has no effect on the -port_list switch. For more information on using -unrolled with -event_list and -event_data, refer to the descriptions of those switches below.

- -port_list

  An optional switch that specifies to return the list of ports to which values are written or from which values are read in the specified pattern set.

- -event_list

  An optional switch that specifies to return the list of all event identifiers of the pattern set.

  When the -event_list switch is used without the -unrolled switch, the returned list contains identifiers of events of type **vector**, **scan**, **reset**, **loop**, **goto_state** or **nop**.

  When the -event_list switch is used with the -unrolled switch, the returned list only contains identifiers of events of type **vector**, **loop** and **nop**. Events of type **scan**, **reset** and **goto_state** are unrolled or broken into the appropriate **vector** events.

  See "Event Types" below for details of all the event types.

- -event_data *event_id*

  An optional switch and string pair that specifies to return a list of key/value pairs of event data. *event_id* must be a string from the list returned by **get_pattern_set_data -event_list**.

You must first request the list of event identifiers from the pattern set, then you can use the event identifiers to get the pattern set data event by event.

Note

It is your responsibility to ensure that the -unrolled switch is used consistently. The event identifiers obtained by means of **get_pattern_set_data –event_list** only make sense in **get_pattern_set_data –event_data *event_id***, and the event identifiers obtained by means of **get_pattern_set_data –unrolled –event_list** only make sense in **get_pattern_set_data –unrolled –event_data *event_id***. As the event_ids are not guaranteed to be mutually exclusive, you must be careful to use –unrolled consistently.

The return value of **get_pattern_set_data pattern_set_name -event_data *event_id*** is always a list with an even number of elements, in which the first, third, fifth ... elements represent keys and the second, fourth, sixth ... elements represent values. This convention allows the conversion into an associative TCL array by means of the following command:

```
array set <variable name> <even-sized list>
```

TCL allows you to iterate on a list using several variables. This is useful to iterate on the results of **get_pattern_set_data**. For example, you can do the following:

```
foreach {key value} [get_pattern_set_data test -event_data] { … }
```

Some of the key/value pairs are guaranteed to appear in the result, others appear only if the related values exist in the retargeted patterns. The entries in the tables are flagged as “mandatory” or “optional” to reflect this.

Event Types

A **vector** event represents a single parallel vector, the stimulus data of the primary inputs (or bidirectional ports) of the design and the response data of primary outputs (or bidirectional ports).

A **scan** event represents one scan load/unload.

A **reset** event represents a global reset (like it would be invoked by e.g. applying a ‘0’ to the TRST port of a TAP controller).

A **loop** event represents a certain idle time. The **vector_count** in the event data refers to tester periods, potentially rounded towards the next integer.

A **goto_state** event represents the transition from one TAP state to another one. The paths for those transitions are the same as the default paths for TAP state transitions in SVF.

A **nop** event (“no operation”) represents the absence of any action. It is only required to hold the information about notes (iNotes) that have been added after the last actual action.

The type of the event can be derived from the identifier names. An event of type **vector** has an identifier named **vector *number***, where ***number*** is an increasing index. An event of type **scan** has an identifier named **scan *number*** and so on.

The following is an example of an event list:

```
{reset0 vector0 scan0 scan1 loop0 scan2 loop1 scan3 vector1}
```

Table 7-11. Events of type "vector"

Key	Value
port_values (mandatory)	String consisting of the characters 'U', 'D', '0', '1', 'X', 'Z', 'M', 'P' and '–'. The order is the same as in the port list (returned by get_pattern_set_data -port_list) The meaning of the characters is as follows: D expect 0 (hint: "down") U expect 1 (hint: "up") M expect Z (hint: "middle" or "median") X do not expect any specific value L drive 0 (hint: "low") H drive 1 (hint: "high") Z drive Z P pulse (only appears in combination with TCK) – ScanIn port during scan load D, U, X, L, H and Z are the same as in SVF. IJTAG adds P, M and – to the list of possible states
write_var.port_list (optional)	Names of the primary input ports (or bidirectional ports) which the write_var.name_list and write_var.inversion_list specifications refer to. Each port can occur only once
write_var.name_list (optional)	Names of iWriteVars associated with the ports. One element per port, same order as in the write_var.port_list
write_var.inversion_list (optional)	Specifies the inversion of the iWriteVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the write_var.port_list
read_var.port_list (optional)	Names of the primary output ports (or bidirectional ports) which the read_var.name_list and read_var.inversion_list specifications refer to. Each port can occur several times, because an output port can be used to observe several targets at the same time
read_var.name_list (optional)	Names of iReadVars associated with the ports. One element per port, same order as in the read_var.port_list.
read_var.inversion_list (optional)	Specifies the inversion of the iReadVars. The elements of the list can be 0 (no inversion) or 1 (inversion). The list must contain the same number of elements as the read_var.port list

Table 7-11. Events of type “vector”

Key	Value
notes.text_list (optional)	List of notes preceding this event
notes.type_list (optional)	List of the types of the notes. The list has the same number of elements as notes.text_list. The elements can be either “user” or “process”. “user” notes are those which have been generated as a result of the iNote command. “process” notes are notes generated by the tool

Table 7-12. Events of type “scan”

Key	Value
port_values write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in “vector” If the port_values contain expected values other than ‘X’, this indicates that the comparison must be done in each cycle of the scan load
scan_interface (mandatory)	Name of the scan interface (as specified in ICL)
scan_type (mandatory)	“DR” or “IR”. Always “DR” for designs without TAP
start_state (mandatory)	Start state of the TAP controller. Can be one of IDLE, DRPAUSE or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, “DRPAUSE” means that there is no capture in this scan load, while “IDLE” means that there is a capture
end_state (mandatory)	End state of the TAP controller. Can be one of IDLE, DRPAUSE or IRPAUSE. For non-TAP designs, it can be IDLE or DRPAUSE. In this case, “DRPAUSE” means that there is no update in this scan load, while “IDLE” means that there is an update
chain_list (mandatory)	List of identifiers of active chains. If the active ScanInterface does not contain Chain statements, then the list consists of only one single element, which is called “default”
<chain_id>.length (mandatory)	Length of the scan chain

Table 7-12. Events of type “scan”

Key	Value
<chain_id>.offset (mandatory)	The number of overshift cycles, i.e. the difference between the number of shift cycles and the length of the chain
<chain_id>.inversion (mandatory)	The chain inversion between ScanInPort and ScanOutPort
<chain_id>.si_values (mandatory)	Scan-in value. A string consisting of ‘0’ and ‘1’ as wide as length+offset
<chain_id>.so_values (mandatory)	Scan-out value. A string consisting of ‘0’, ‘1’ and ‘X’ as wide as length+offset
<chain_id>.write_var.cycle_list (optional)	List of cycles which the <chain_id>.write_var.name_list and <chain_id>.write_var.inversion_list refer to
<chain_id>.write_var.name_list (optional)	List of iWriteVars, consistent with cycle_list, analogously defined as write_var.name_list
<chain_id>.write_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as write_var.inversion_list Note: the inversions take the scan-in inversions into account, such that it can be derived directly what has to be applied to the scan-in port
<chain_id>.read_var.cycle_list (optional)	List of cycles which the <chain_id>.read_var.name_list and <chain_id>.read_var.inversion_list refer to
<chain_id>.read_var.name_list (optional)	List of iReadVars, consistent with cycle_list, analogously defined as read_var.name_list
<chain_id>.read_var.inversion_list (optional)	List of inversions, consistent with cycle_list, analogously defined as read_var.inversion_list Note: the inversions take the scan-out inversions into account, such that it can be derived directly what has to be expected at the scan-out port
<chain_id>.register.name_list	The names of the ICL ScanRegisters in the correct order from scan-in to scan-out
<chain_id>.register.length_list	The list of register lengths in the order from scanin to scan-out. The elements in this list can be mapped uniquely to the elements in the list which is referenced by <chain_id>.register.name_list

Table 7-13. Events of type “reset”

Key	Value
notes.text_list notes.type_list	Same as in “vector”
sync (mandatory)	“On” or “Off”. Specifies whether a synchronous reset shall be enforced
type (mandatory)	“user” or “process”. “user” indicates the usage of iReset, whereas “process” indicates a reset generated by the tool, e.g. on behalf of the open_pattern_set command (if used without no_initial_ireset)

Table 7-14. Events of type “loop”

Key	Value
port_values write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in “vector” If the port_values contain expected values other than ‘X’, this indicates that the comparison must be done in each cycle of the loop
vector_count (mandatory)	Duration in terms of tester periods, rounded towards the next integer

Table 7-15. Events of type “goto_state”

Key	Value
port_values write_var.port_list write_var.name_list write_var.inversion_list notes.text_list notes.type_list	Same as in “vector” If the port_values contain expected values other than ‘X’, this indicates that the comparison must be done in each cycle of the transition
start_state (mandatory)	IDLE, DRPAUSE or IRPAUSE
end_state (mandatory)	IDLE, DRPAUSE or IRPAUSE

Table 7-16. Events of type “nop”

Key	Value
notes.text_list notes.type_list	Same as in “vector”

Example

The following example illustrates the usage of the get_pattern_set_data command.

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl data/icl/*.icl
SETUP> set_current_design chip
SETUP> add_clocks ClkA -free_running -period 10ns
SETUP> set_system_mode analysis
ANALYSIS> source data/pdl/raw1.iprocs
ANALYSIS> open_pattern_set test1 -tester_period 100ns
ANALYSIS> iCall block1_I1.raw1_I1.run_testa
ANALYSIS> close_pattern_set
ANALYSIS> get_pattern_set_data test1 –port_list

tck tdi tms trst tdo
ANALYSIS> get_pattern_set_data test1 –event_list

reset0 scan0 scan1 scan2 scan3 scan4 scan5 scan6 loop0 scan7
ANALYSIS> array set pd [get_pattern_set_data test1 –event_data scan5]
ANALYSIS> puts $pd(chain_list)

default
ANALYSIS> puts $pd(default.length)

16
ANALYSIS> puts $pd(default.read_var.name_list)

block1_I1.raw1_I1.go block1_I1.raw1_I1.done
ANALYSIS> puts $pd(default.read_var.cycle_list)

7 8
ANALYSIS> puts $pd(default.so_values)

XXXXXXX00XXXXXXX
ANALYSIS> exit –f
```

Related Commands

open_pattern_set

close_pattern_set

get_pattern_set_list

Context: patterns -ijtag

Mode: analysis

Usage

get_pattern_set_list

Description

Returns a Tcl list of all existing and closed pattern sets in the order of their creation.

The currently open pattern set is not part of the list.

Arguments

- None

Return Value

A Tcl list of all existing and closed pattern sets in the order of their creation.

Example

The following example sets the patlist variable to the set of all existing pattern sets.

```
set patlist [get_pattern_set_list]
```

Related Commands

close_pattern_set
get_open_pattern_set
open_pattern_set
report_pattern_sets
reset_open_pattern_set

get_pattern_set_option

Context: patterns -ijtag

Mode: analysis

Usage

get_pattern_set_option [pattern_set_name] {**-timeplate_name | -tester_period | -cycles | -tck_ratio | -initial_ireset | -active_scan_interfaces | -network_end_state | -tap_start_state | -tap_end_state | -saved**}

Description

Returns the value of the requested option for the specified pattern_set_name.

Arguments

- pattern_set_name

  Specifies an optional pattern set name from which to introspect the specified option. When not specified, the currently open pattern set is introspected.

- **-timeplate_name**

  Returns the name of the optional timeplate that was used when opening the pattern set. Returns null if the open_pattern_set -timeplate option was not specified; in this case the default timeplate is used. See open_pattern_set for a description of the default timeplate.

- **-tester_period**

  Returns the tester period used by the pattern set.

- **-cycles**

  Returns the complete number of tester cycles required to execute the pattern set on a tester.

- **-tck_ratio**

  Returns the tck_ratio value of the pattern set.

- **-initial_ireset**

  Returns 1 when the pattern set was opened without the -no_initial_ireset option; returns 0 when the option was used.

- **-active_scan_interfaces**

  Returns the names of the active scan interfaces.

- **-network_end_state**

  Returns keep, initial, or reset based on the value used with the close_pattern_set -network_end_state option.

- **-tap_start_state**

  Returns the expected TAP start state. The possible values are: “IDLE”, “DRPAUSE”, “IRPAUSE” or “any”.

- **-tap_end_state**

  Returns the established TAP end state. The possible values are: “IDLE”, “DRPAUSE”, or “IRPAUSE”. An error message is issued if you try to get the TAP end state of an open pattern set.

- **-saved**

  Returns 1 when the pattern set was written out using at least one write_patterns command. Returns 0 otherwise.

Return Value

The value of the requested option.

Example

In the following example, the total number of cycles for pattern set P1 is stored in the Tcl variable V1.

```
set V1 [get_pattern_set_option P1 -cycles]
```

Related Commands

open_pattern_set

close_pattern_set

get_pins

Context: all contexts

Mode: all modes

Usage

get_pins [name_patterns] [-of_instances *instance_objects* |
-below_instances *instance_objects* | -of_nets *net_objects* |
-of_gate_pin *gate_pin_objects*| -of_pseudo_ports *pseudo_port_objects*]
[-hierarchical] [-filter *attribute_equation*] [-direction input | output | inout]
[-regexp] [-nocase] [-silent]

Description

Returns a collection of all hierarchical pins instantiated relative to the current design that match the specified name_patterns list and subject to filtering imposed by the other optional options.

When the name_patterns list is omitted, all pins are returned subject to filtering imposed by the other optional options.

Arguments

- *name_patterns* | *leaf_instance/leaf_name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of pins. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all pins. The name patterns are not allowed to include more than one slash which is the hierarchy separator symbol. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a complete description of the pattern syntax when the -regexp option is used and not used.

  This argument cannot be specified with either the -of_nets or -of_gate_pin arguments.

- -of_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for pins on the specified *instance_objects*. When this option is specified, the optional name_patterns must be leaf_name patterns that are checked against the leaf name of the pins. An error is generated if the specified name_patterns contain more than one level of hierarchy when the -of_instances option is used. You could traverse the module_name attribute of the instance objects and look up the instantiated modules but this method is more direct and efficient and automatically deals with libraries and architectures.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for pins below the specified *instance_objects.* Specified name_patterns are searched relative to the specified *instance_objects* when this option is used.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated to *net_objects*. Those are the pins having the exact same name as the nets and reside on the instances in which the nets are in. You cannot use a *name_patterns* list when using this option.

- -of_gate_pin *gate_pin_objects*

  An optional switch and value pair that constrains the command to return the pin objects associated to the specified *gate_pin_objects*. Given, a collection or list of gate_pin objects, the option is used to map the gate_pin object to the corresponding hierarchical pin objects. Any gate_pin object not having an associated hierarchical pin object is discarded. You cannot use a *name_patterns* list when using this option.

- -of_pseudo_ports *pseudo_port_objects*

  An optional switch and value pair that constrains the command to return the design pins associated with the specified pseudo-ports (user-added primary inputs or outputs).

- -hierarchical

  An optional string that specifies to perform pattern matching relative to all instances found below the current design. When this option is specified, the name_patterns argument cannot include more then one slash which are hierarchy separator symbols.

- -direction input | output | inout

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the pin object. When omitted, pins with any direction are considered.

- -filter *attribute_equation*

  An optional switch and string pair that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the pin object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operator use simple wildcard matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for pin names matching *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a

probable mistake when specifying patterns. However, it is possible the pin targeted by name_patterns disappeared due to a change in a previous editing step. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

In the following example, the collection of all pins is filtered based on an attribute_equation of user-defined attributes.

get_pins -filter {myType1 || myType2 == "ABC"}

```
{u1/Y u4/u4/A0 u7/D}
```

Example 2

In the following example, the collection of all pins matching a list of hierarchical name patterns is returned. The names u1/Y and u*/Q are only searched relative to the current design. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for examples of pattern matching with regular expressions.

get_pins {u1/Y u*/Q}

```
{u1/Y u2/Q u7/Q}
```

Example 3

In the following example, the pins with leaf_name matching Q* and found on instances of module DFF are returned

```
get_pins Q* -of_instances [get_instances -of_modules DFF]
{u2/reg1/Q u2/reg1/QB u3/reg5/Q}
```

Example 4

In the following example. the collection of all pins matching a leaf instance_name/leaf pin name pattern is returned. The tool searches for the leaf instance name u* in all instances found below the current design, and then searches for the leaf pin names matching Y* on those instances.

get_pins u*/Y* -hierarchical

```
{u1/Y u3/Y u4/u1/Y u4/u4/Y u4/u5/u1/Y u5/u1/Y u5/u3/Y u5/u4/Y u6/Y}
```

Example 5

This example return the pin names of the specified pseudo_port objects:

get_pins -of_pseudo_ports {user_pi1 user_pi2}

```
{u1/Y u3/Y u4/Y}
```

Related Commands

create_pin
delete_pins
get_common_parent_instance
get_modules
get_common_parent_instance
get_nets
get_icl_ports

get_ports

Context: all contexts

Mode: all modes

Usage

get_ports [*name_patterns*]
[-of_module *module_object* | -of_nets *net_objects* |
-of_pins *pin_objects* | -of_gate_pins *gate_pin_objects*]
[-direction input | output | inout]
[-of_type real | pseudo | all]
[-filter *attribute_equation*] [-regexp] [-nocase] [-silent]

Description

Returns a collection of all hierarchical ports on a given module that match the specified name_patterns list and subjected to filtering imposed by the other options.

You can have any number of wildcards or regular expressions in the name patterns.

Arguments

- *name_patterns*

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of ports. If no name_patterns are specified, the tool searches all ports.

- -of_module *module_object*

  An optional switch and value pair that constrains the command to return only the ports on the module specified by module_object. The module_object is a single module name, or a collection of one module object as returned by the get_modules command.

- -of_nets *net_objects*

  An optional switch and value pair that constrains the command to return the port objects associated to net_objects. These are the ports having the exact same name as the nets and residing on the current design. You cannot use a name_patterns list when using this option.

- -of_pins *pin_objects*

  An optional switch and value pair that constrains the command to return the ports defined on the modules that correspond to the pins specified by pin_objects.

  You can further constrain the command to return only Pseudo_ports or only real Ports using "-of_type pseudo | real"; if you do not specify the -of_type switch, by default, the tool returns all real ports (-of_type real). If you use this option in conjunction with "-of_type pseudo", the tool maps the pin that was added as part of the pseudo_port (user-added PI/PO) to the name of the associated pseudo_port. See Example 4. If you added a user-specified primary input or output (pseudo_port) without specifying the pin name using add_primary_inputs -pin_name, the name of the pseudo_port may not always match the

name of the pin specified; in this case, you can use this switch to find the name of the associated pseudo_port.

You cannot use this switch with the "-of_type all" option. You also cannot use this switch with a name_patterns list.

- -of_gate_pins *gate_pin_objects*

  An optional switch and value pair that constrains the command to return the ports on the current design associated to the gate_pin object. Any gate_pin object not associated to a port object on the current design is discarded. You cannot use a name_patterns list when using this option.

- -direction input | output | inout

  An optional switch and literal pair that specifies to filter results based on the value of the direction attribute on the port object. When omitted, ports with any direction are considered.

- -of_type real | pseudo | all

  An optional switch and literal pair that specifies to filter results based on the specified type of the port object. When omitted, only real design ports are considered. The option types have the following meaning:

  real — Only real design ports (Port object type) are included in the returned collection.

  pseudo — Only user-added pseudo ports (Pseudo_port object type) are included in the returned collection.

  all — Both real (design) and pseudo (user-added) ports are included in the returned collection.

- -filter *attribute_equation*

  An optional switch and string that specifies to filter results based on the expression specified by the attribute_equation string. The attributes used within the equation must exist for the port object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format. The =~ and !~ matching operators use simple wild card matching versus Posix extended regular expression based on the presence of the -regexp option.

- -regexp

  An optional switch that directs the tool to interpret the value of the *name_patterns* argument as a real regular expression instead of as a simple wildcard pattern. See section "Glob and Regular Expression Pattern Matching Syntax" on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that directs the tool to perform case-insensitive pattern matching when looking for ports matching the *name_patterns*.

- -silent

  An optional switch that suppresses the warning message normally generated when the -filter argument returns an empty collection. The warning message is intended to indicate a probable mistake when specifying patterns. However, it is possible the module targeted by the name_patterns disappeared due to a change in a previous editing step. You can use the -silent switch to manage the case in which an empty collection is returned. This -silent option has no effect on any other error reporting.

Example 1

This example looks for ports on the current design whose name match I* or O*

get_ports {I* O*}

```
{I1 I2 I3 I4 I5 I6 I7 I8 O1 O2 O3}
```

Example 2

This example looks for scalar ports having one or two characters on module ModA.

get_ports {[[:alnum:]]{1,2}} -regexp -of_module ModA

```
{ab a b}
```

Example 3

This example return ports based on the option specified for the -of_type switch:

add_primary_input -internal inst1/Y inst1/Z -pin_name user_pi1
add_primary_input -internal inst2/Y -pin_name user_pi2
get_ports

```
{port1 port2 port3}
```

get_ports -of_type real

```
{port1 port2 port3}
```

get_ports -of_type pseudo

```
{user_pi1 user_pi2}
```

get_ports -of_type all

```
{port1 port2 port3 user_pi1 user_pi2}
```

get_pins -of_pseudo_ports user_pi1

```
{inst1/Y inst1/Z}
```

get_pins -of_pseudo_ports user_pi2

```
{inst2/Y}
```

get_pins -of_pseudo_ports {user_pi1 user_pi2}

```
{inst1/Y inst1/Z inst2/Y}
```

Example 4

This example returns the pseudo_ports associated with the user-specified primary inputs and primary outputs based on the option specified for the -of_type switch:

add_primary_input -internal inst1/Y inst1/Z -pin_name user_pi1
add_primary_input -internal inst2/Y

get_ports -of_pins inst1/Y

```
{Y}
```

get_ports -of_pins {inst1/Y inst1/Z} -of_type real

```
{Y Z}
```

get_ports -of_pins inst1/Y -of_type pseudo

```
{user_pi1}
```

get_ports -of_pins inst2/Y -of_type real

```
{Y}
```

get_ports -of_pins inst2/Y -of_type pseudo

```
{inst2/Y}
```

get_ports -of_pins {inst1/Y inst1/Z inst2/Y} -of_type pseudo

```
{user_pi1 user_pi1 inst2/Y}
```

Related Commands

create_port

delete_ports

get_procfile_name

Context: all contexts

Mode: all modes

Usage

get_procfile_name

Description

Returns the name of the procfile that was defined using the set_profile_name command.

Arguments

- None

Return Value

The name of the procfile as it was specified to the set_procfile_name command.

Example

In the following example, the path to the specified procfile name is stored in a Tcl variable.

```
set proc_file [get_procfile_name]
```

Related Commands

set_procfile_name

get_resource

Context: unspecified, all contexts

Mode: all modes

Usage

get_resource ***resource_list***

Description

Returns true when all resources in the specified resource_list are available.

You use this command inside your Tcl procs to check for the availability of resources and take appropriate action when they are missing. For example, you can choose to issue the create_flat_model command if your command depends on it and it is not yet available.

Arguments

- ***resource_list***

  A Tcl list of strings that contains any of the following resource names:

  design_is_elaborated

  Used to check that the hierarchical data model has been created using the set_current_design command.

  design_modules_exist

  Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the get_modules command.

  flat_model_exists

  Used to check that the flat data model is available. The flat data model was created by either the create_flat_model command or by the “set_system_mode analysis” command.

  icl_is_elaborated

  Used to check that the ICL hierarchical data model exists. It exists after the set_current_design command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

  icl_modules_exist

  Used to check that at least one ICL module exists in memory and can be accessed using the get_icl_modules command.

  rtl_mode_enabled

  Used to check that the -rtl option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

Example

The following example checks that both design and ICL modules exist.

```
if {![get_resource {design_modules_exist icl_modules_exist}]} {
   return -code error "you must first read in design and icl modules \
      before executing this command"
}
```

Related Commands

create_flat_model
read_icl
read_verilog
read_vhdl
set_context
set_current_design
set_system_mode

get_scratch_directory

Context: unspecified, all contexts

Mode: all modes

Usage

get_scratch_directory

Description

Returns the absolute path to the *scratch* directory.

The scratch directory is created in the temporary *.tessent.tmp* directory. The *.tessent.tmp* directory is created at tool invocation and deleted when the tool exits. The directory is created in the current working directory if write permissions are open; otherwise, it is created in */tmp*.

You can explicitly specify the location in which the *.tessent.tmp* directory is created by setting the TESSENT_TMP_LOCATION environment variable. If the location specified by this environment variable does not exist, is not a valid directory, or does not have write permission, the tool creates the *.tessent.tmp* directory in the default location which is the current working directory.

Arguments

None

Example

The following example displays the location of the scratch directory:

get_scratch_directory

```
/wv/dft/.tessent.tmp.rainbow.26857/scratch
```

get_simulation_context_list

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

get_simulation_context_list [-predefined | -user_defined]

Description

Returns the available simulation contexts in a Tcl list.

The get_simulation_context_list command returns all simulation contexts or returns only predefined or user-defined. If you issue the command without arguments, the command returns all simulation contexts. Note that for predefined simulation contexts, this command returns only the simulation contexts for which simulation data currently exists. This requires that the tool first read a procedure file.

Arguments

- -predefined

  An optional switch that specifies predefined simulation contexts, which are listed in Table 7-40 on page 1648.

- -user_defined

  An optional switch that specifies user-defined simulation contexts.

Example

The following example shows the Tcl list of values returned by the get_simulation_context_list command:

```
ANALYSIS> set_current_simulation_context stable_load_unload
ANALYSIS> add_simulation_context myContext1
ANALYSIS> add_simulation_context myContext2
ANALYSIS> get_simulation_context_list
{ stable_after_setup stable_load_unload stable_capture stable_shift myContext1 myContext2 }

ANALYSIS > get_simulation_context_list -user_defined
{  myContext1 myContext2 }

ANALYSIS > get_simulation_context_list -predefined
{ stable_after_setup stable_load_unload stable_capture stable_shift }
```

Related Commands

add_simulation_context
copy_simulation_context
delete_simulation_contexts
get_current_simulation_context
get_simulation_value_list
report_simulation_contexts
set_current_simulation_context

get_simulation_value_list

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

get_simulation_value_list ***obj_spec***

Description

Returns a Tcl list of the simulation values on the specified gate_pin objects.

The get_simulation_value_list command returns the simulation values on the specified gate_pin objects in the current simulation context. The simulation value of a single gate_pin is either X, 0, 1, or Z.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects or pins that map to gate_pin objects (on library cell boundaries or preserved design boundaries).

Example

The following example shows that the get_simulation_value_list command is equivalent with querying the simulation_value attribute. However, note that get_attribute_value_list command requires a Tcl list of gate_pins in order to work correctly, as shown in the example:

```
> set_current_simulation_context stable_after_setup
> get_simulation_value_list {pi clk an_2/A0 an_2/Y}
1 X 1 X

> get_attribute_value_list [get_gate_pins {pi clk an_2/A0 an_2/Y}] -name simulation_value
1 X 1 X
```

Related Commands

add_simulation_force
delete_simulation_forces
report_simulation_forces

get_single_name

Context: unspecified, all contexts

Mode: all modes

Usage

get_single_name ***object_spec*** [-remove_escaping] [-silent]

Description

Returns a string with the name of the element specified by the object_spec argument.

The -remove_escaping argument removes the \ and the trailing spaces in an escaped identifier. Many legacy commands such as add_display_instance require the instance name to have the escape character stripped from its name. However, these legacy commands do accept collections as inputs in cases where the name format is not important. See the examples for an illustration of this behavior

If the object in *object_spec* does not exist, an error message displays unless the -silent switch is specified. If more than one object exists in object_spec, an error message displays unless the -silent switch is specified.

Arguments

- ***object_spec***

  A required string that specifies an object name or a single-element collection.

- -remove_escaping

  An optional switch that specifies to strip out the \ and the trailing spaces in escaped identifiers. This option is useful to interact with legacy commands and the collection is not directly passed as input to them. See Example 3 for an illustration of this usage.

- -silent

  An optional switch that specifies to suppress error messages if the object in *object_spec* does not exist.

Example 1

The following example prints out all of the modules in the database.

```
foreach_in_collection elem [get_modules] {
   puts "Module: [get_single_name $elem]"
}

Module: modb
Module: modc
Module: modd
Module: top
Module: GND
...
```

Example 2

The following example generates an error because the collection of pins returned by the get_pins commands contains more than one element.

get_single_name [get_pins u1/i*]

```
//  Error: Collection contains more than one element.
```

Example 3

The following example uses the -remove_escaping argument to strip out the slash "\" from an escaped identifier.

get_single_name {\x[0] } -remove_escaping

```
x[0]
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
get_common_parent_instance
get_icl_pins
get_modules
get_icl_ports
get_name_list

get_system_mode

Context: unspecified, all contexts

Mode: all modes

Usage

get_system_mode [-previous]

Description

Returns the system mode.

Possible returned values are: setup, analysis, and insertion.

Arguments

- -previous

  An optional switch that specifies to return the previous system mode.

Example

The following example returns the current system mode and the previous system mode. This enables you to know that you came to the insertion mode through the analysis mode.

get_system_mode

```
insertion
```

get_system_mode -previous

```
analysis
```

Related Commands

set_system_mode

get_test_end_icall_list

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

Mode: all modes

Usage

get_test_end_icall_list

Description

Returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list containing the iProc and its arguments.

For each iCall added to the test_end procedure by the set_test_end_icall command, the get_test_end_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Example

The following example returns the iCalls added to the test_end procedure by the set_test_end_icall commands. Each iCall is a list consisting of the ICLinstance path, the name of the iProc, and the arguments to the iProc.

set_test_end_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off"

set_test_end_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off" -append

get_test_end_icall

```
{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Commands

get_test_setup_icall_list
iCall
set_test_end_icall
set_test_setup_icall

get_test_setup_icall_list

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

Mode: all modes

Usage

get_test_setup_icall_list

Description

Returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list containing the iProc and its arguments.

For each iCall added to the test_setup procedure by the set_test_setup_icall command, the get_test_setup_icall_list command returns a list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

Arguments

None

Example

The following example returns the iCalls added to the test_setup procedure by the set_test_setup_icall commands. Each iCall is a list consisting of the ICLinstance path, the name of the iProc, and the arguments to the iProc.

set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off"

set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off" -append

get_test_setup_icall

```
{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Related Commands

get_test_end_icall_list
iCall
set_test_end_icall
set_test_setup_icall

get_tool_info

Context: unspecified, all contexts

Mode: all modes

Usage

get_tool_info -name | -version | -cpu_time [ms | s | m | h | d] |
-memory [b | kb | mb | gb] | -hostname

Description

Returns information about the current tool and the current run as a Tcl result.

You must specify exactly one switch.

Arguments

- -name

  A switch that specifies to return the name of the current tool.

- -version

  A switch that specifies to return the version of the current tool.

- -cpu_time [ms | s | m | h | d]

  A switch that specifies to return the user time of the process. By default, this command returns the time in seconds, but you can optionally specify one of the other units. The possible units are:

  ms — Specifies the time unit is milliseconds.

  s — Specifies the time unit is seconds.

  m — Specifies the time unit is minutes.

  h — Specifies the time unit is hours.

  d — Specifies the time unit is days.

- -memory [b | kb | mb | gb]

  A switch that specifies to return the memory usage. By default, this command returns the usage in bytes, but you can optionally specify one of the other units.

  b — Specifies the memory usage unit is bytes.

  kb — Specifies the memory usage unit is kilobytes.

  mb — Specifies the memory usage unit is megabytes.

  gb — Specifies the memory usage unit is gigabytes.

- -hostname

  A switch that specifies to return the hostname.

Example

The following examples return information about the current tool:

get_tool_info -name

```
tessent_shell
```

get_tool_info -version

```
2012.4
```

get_tool_info -cpu_time

```
2.773
```

get_tool_info -memory

```
228343808.0
```

get_tool_info -hostname

```
galactica
```

Related Commands

report_licenses

report_resources

get_trace_flat_model_option

Context: all contexts

Mode: setup, analysis

Usage

get_trace_flat_model_option [-controllability] [-latch] [-tag_condition] [-stop_condition] [-stop_at_first_tag] [-max_levels] [-map_tag_to_design_module_boundary]

Description

Introspects the default behavior of the trace_flat_model command.

You can only specify one option at a time. The tool generates an error if more than one option is specified in a single command execution.

Arguments

- -controllability

  An optional switch that returns the current setting of the controllability option. Possible return values are:

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion.

  constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

  unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

  connected—With this value, gates are always traversed when reached.

- -latch

  An optional switch that returns the current setting of the latch option. Possible return values are:

  normal — A latch is traced through only if the simulated value in stable_after_setup makes the latch satisfy the specified controllability requirements.

  transparent — All latches are traced through regardless of the value at the latch enable port.

- -tag_condition

  An optional switch that returns the current setting of the tag condition option. The returned value is an attribute expression. For more details about filtering attribute equation formats, refer to “Attribute Filtering Equation Syntax” on page 105.

- -stop_condition

  An optional switch that returns the stop condition for the trace. The returned value is an attribute expression.

- -stop_at_first_tag

  An optional switch that returns the current setting of the -stop_at_first_tag option. This option specifies whether to interrupt tracing when the first tag object with the specified controllability is found. Possible return values are: {ON | OFF}

- max_levels

  An optional switch that returns the current setting of the max_levels option. This option specifies the maximum level of gates to trace. Possible return values are: unlimited or a number.

- -map_tag_to_design_module_boundary

  An optional switch that returns the current setting of the map_tag_to_design_module_boundary option, which specifies whether to map the tag points to gate_pins outside of library cells. Possible return values are: {ON | OFF}

Example 1

The following example returns the current setting for the -controllability option.

```
get_trace_flat_model_option -controllability
```

Example 2

The following example returns the current setting for the -tag_condition option.

```
get_trace_flat_model_option -tag_condition
```

Example 3

The following example returns the current setting for -stop_condition option.

```
get_trace_flat_model_option -stop_condition
```

Example 4

The following example returns the current setting for the -stop_at_first_tag option.

```
get_trace_flat_model_option -stop_at_first_tag
```

Example 5

The following example returns the current setting for the -max_levels option.

```
get_trace_flat_model_option -max_levels
```

Example 6

The following example returns the current setting for the -latch option.

```
get_trace_flat_model_option -latch
```

Related Commands

set_trace_flat_model_options

trace_flat_model

get_transcript_style

Context: unspecified, all contexts

Mode: all modes

Usage

get_transcript_style

Description

Returns the keyword that specifies the style in which the tool transcripts commands.

The get_transcript_style command returns one of four keywords previously set with the set_transcript_style command. Possible return values are: off, full, tool_commands_only, and input_only.

Note that issuing the set_transcript_style command without an argument prints the style. Issuing get_transcript_style returns a value that you can assign to a variable for later use (see example below).

Arguments

None

Example

The following example stores the current transcript style in a variable, turns off transcripting, then later restores the original transcript style:

```
set old_transcript_style [get_transcript_style]
set_transcript_style off

... # do something with transcripting turned off

set_transcript_style $old_transcript_style
```

Related Commands

set_logfile_handling

set_transcript_style

get_tsdb_list

Context: all contexts

Mode: all modes

Usage

get_tsdb_list

Description

Returns a Tcl list of directory paths, relative to the current working directory, corresponding to all opened TSDB directories.

When the default or specified TSDB output directory is already created, it is returned in the list even if there was no explicit open_tsdb command for it.

Arguments

None

Examples

The following example shows the use of the get_tsdb_list command before and after closed some of them.

```
> get_tsdb_list
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
> close_tsdb [list ${tsdb_home}/corea.tsdb \
                   ${tsdb_home}/coreb.tsdb ]
> get_tsdb_list
/home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

open_tsdb

close_tsdb

set_tsdb_output_directory

get_tsdb_output_directory

get_tsdb_output_directory

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Usage

get_tsdb_output_directory

Description

Returns the directory used by the process_dft_specification command which is the directory that has been previously set by the set_tsdb_output_directory command.

The TSDB output directory defaults to *./tsdb_outdir* until the set_tsdb_output_directory command has been invoked. When either the get_tsdb_output_directory or set_tsdb_output_directory command is invoked, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it is not.

The process_dft_specification command is currently the only command that makes use of the TSDB directory.

Arguments

None

Example 1

The following example stores the TSDB output directory into a Tcl variable called tsdb_outdir.

set tsdb_outdir [get_tsdb_output_directory]

Example 2

The following example shows a possible error than can be generated when invoking the command.

set_tsdb_output_directory dir1
exec “rm -r dir1; touch dir1”
get_tsdb_output_directory

```
//  Error: The specified tsdb_output_directory 'dir1' exists but is not a
directory.
//  Error: Directory was either deleted or its write permission changed
between now and the invokation of set_tsdb_output_directory command.
```

Related Commands

process_dft_specification

set_tsdb_output_directory

help

Context: unspecified, all contexts

Mode: all modes

Usage

help [*command_name*] [-manual] [-all]

Description

Lists commands and associated usage syntax.

Specifying the help command without any arguments returns a list of all available commands. Specifying a single command name displays its usage syntax.

Arguments

- *command_name*

  An optional string that specifies the name of the command for which you want help. Entering the first part of a command name returns a list of commands that start with the specified string. The *command_name* can include wildcard characters. For example, you can use an "*" to match zero or more characters at the position of the asterisk in the string.

  You also can use the tab completion method on available commands, which supplies missing characters between underscores. For example, specifying "help se_sy_m" and pressing the Tab key inserts "help set_system_mode" on the command line. Tab completion also works in this same way when invoking any tool command.

  If a single command is returned, its usage syntax also displays.

- -manual

  An optional switch that displays the reference manual description for the specified command. If you use this switch without specifying a command name, the tool opens the Tessent documentation system, giving you access to all the manuals for the product group.

- -all

  An optional switch that displays all command matches, including unavailable commands, which are identified with the "(...unavailable...)" notation. Specifying "help *unavailable_command_name*" displays the reason why a particular command is unavailable.

Example 1

The following example returns the usage syntax for the history command:

help history

```
history [ <count> ] [ -Nonumbers ] [ -Reverse ] [ -Save <savefile> ]
```

Example 2

The following example returns all commands in the current mode that begin with the string *set_design*:

help set_design

```
set_design_include_directories
set_design_macros
set_design_sources
```

Example 3

The following example returns all commands in the current mode that contain the string *design*:

help *design*

```
delete_design
get_current_design
set_current_design
set_design_include_directories
set_design_macros
set_design_sources
write_design
```

Example 4

The following example returns all commands containing the string *design*, including those that are unavailable:

help *design* -all

```
delete_design
find_design_names      (...unavailable...)
get_current_design
set_current_design
set_design_include_directories
set_design_macros
set_design_sources
write_design
```

history

Context: unspecified, all contexts

Mode: all modes

Usage

history [*list_count*] [-Nonumbers] [-Reverse] [-Save *filename*]

Description

Displays a list of previously executed commands.

You can use command line editing to recall a previous command and edit it for submission as a new command.

A leading number precedes each command line in the history list that indicates the order in which the commands were entered.

Arguments

- *list_count*

  An optional integer that specifies for the tool to display only the specified number (*list_count*) of most recent executed commands. If no *list_count* is specified, the tool displays all previously executed commands.

- -Nonumbers

  An optional string that specifies for the tool to display the history list without the leading numbers. This is useful for creating dofiles. The default displays the leading numbers.

- -Reverse

  An optional switch that specifies for the tool to display the history list starting with the most recent command rather than the oldest.

- -Save *filename*

  An optional switch and string pair that specifies the name of a file to which to write the command history. This is useful for creating dofiles.

Example 1

The following command saves the command history into a dofile for later execution.

history -save dofile1.txt

Example 2

The following command displays the history list with leading numbers, starting with the oldest command.

history

```
1  help hist
2  dof instructor/fault.do
```

```
 3  set_system_mode analysis
 4  set_fault_type stuck
 5  add_faults -all
 6  create_patterns
 7  report_statistics
 8  report_faults -class ATPG_UNTESTABLE
 9  analyze_fault /I$20/en -stuck_at 1
10  set_system_mode setup
11  set_system_mode analysis
12  set_fault_type iddq
13  add_faults -all
14  create_patterns
15  report_statistics
16  history
```

iApply

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iApply [-end_in_pause]

Description

Triggers the retargeting of all queued iRead and iWrite commands.

This command causes the retargeting engine to calculate the optimal sequence of scan loads, primary input stimulations, and primary output observations to satisfy all queued iRead targets and iWrite targets, as illustrated in Figure 7-15.

The number of parallel IO time frames and/or Scan time frames used to satisfy all queued up iWrite and iRead commands depends on the number of queued commands and the circuit topology described in the ICL description.

The solution criteria for the iWrite commands is that all specified iWrite values are achieved and present at the end of the iApply time frame. The order in which they become satisfied is unknown and is chosen by the solver in order to minimize the number of clock cycles needed to complete the tasks. The solver is designed to minimize transition on the writable objects and will limit them to a single transition per iApply time frame if the ICL circuit allows it.

If it is not possible to achieve all specified iWrite values at the end of the time frame, the specified iWrite command set is said to have no solution. The iRead object only needs to be observed once anywhere within the time frame. The value is strobed at the first opportunity that it becomes observable and is then ignored if re-observed on subsequent Scan or parallelIO time frames.

If there is no solution for the retargeting task, the iRead targets and iWrite targets are taken out of the queue and the current state of the circuit is retained. An error message appears in this case. If there is a solution, the solution is appended to the opened pattern set. The circuit state is updated to reflect the situation after the actions within the current iApply time frame have been applied.

You can use the optional -end_in_pause switch to help create IJTAG scan chain integrity tests. This switch allows you to observe the same values at the scan output that have been shifted in before. When you don’t use this switch, the shift register content is always overwritten by the capture values before being observed at the scan output.

Figure 7-15. iApply Time Frame

Arguments

- -end_in_pause

  An optional switch that helps you to create IJTAG scan chain integrity tests. The -end_in_pause switch changes the handling of the last scan load of the iApply and the first scan load of the next iApply.

  Note that the following explanation refers to states of a finite state machine (FSM) of a IEEE 1149.1 (JTAG) TAP controller. “IR” is instruction register, and “DR” is data register. For a complete explanation about how the TAP controller operates, refer to the IEEE 1149.1 standard document.

  When you use the -end_in_pause switch, and the last scan load is a “TAP scan load” (capture, shift and update activity is controlled by a TAP state machine), then the TAP state machine ends in the pause-DR or pause-IR state (that is, the tool performs no update-DR/update-IR). The first scan load of the next iApply starts from there and performs another scan load (obviously without capture-DR/capture-IR) before returning to RTI (run test/idle).

  When you use the -end_in_pause switch, and the last scan load is an ordinary scan load (capture, shift, and update activity is controlled by the appropriate CaptureEnPort, ShiftEnPort, UpdateEnPort), then the “update” of the last scan load of the iApply is skipped. The first scan load of the next iApply skips the “capture.” For more information about the CaptureEnPort, ShiftEnPort, UpdateEnPort keywords, refer to “ICL and PDL Modeling” in the *Tessent IJTAG User's Manual*.

  You can use this switch only when the iWrite targets are all part of ScanRegisters. Those ScanRegisters must be configurable to appear as one concatenated scan chain.

Return Value

None

Example 1

In the following example, two iWrite and two iRead commands are queued up when the iApply command is issued. The solver starts from the current state of the ICL circuit and looks for a series of ParallelIO and Scan time frames to have *u1.ina* equal to "*11*" and *u1.inb* equal to "*0*" at the end of the iApply time frame, and to have observed at least once the u1.out1 and u1.out2 objects.

The number of ParallelIO and Scan time frames needed to achieve the objective depends on the ICL network. If both *u1.ina* and *u1.inb* shared a common source in ICL, then the given PDL could not be satisfied because it would be impossible to have *u1.in1* high with *u1.inb* low together at the end of the iApply time frame.

```
iWrite u1.ina 0b11
iWrite u1.inb 0b0
iRead u1.out1
iRead u1.out2 0x7
iApply
```

Example 2

The following example applies a chain integrity test.

```
open_pattern_set chain_pattern
        iWrite myRegister 0b00110011
        iApply -end_in_pause
        iRead myRegister 0b00110011
        iApply
close_pattern_set
```

Related Commands

iRead iWrite

iCall

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iCall [icl_instance_path.]**iproc_name** [arguments …]

Description

Calls an iProc registered against the ICL module associated with the specified *effective_icl_instance_path*.

The *effective_icl_instance_path* is computed by concatenating the following three components:

iProc instance path — Path that exists when running commands specified within an iProc; it is set to the effective_icl_instance_path used in the iCall command that called the iProc. This value is empty when the iCall is not specified within an iProc.

iPrefix path — Path specified as the argument to the iPrefix command since the beginning of the open_pattern_set for commands that are not inside an iProc, or since the beginning of the iProc for commands that are inside an iProc. If no iPrefix command has yet been specified in the current scope, this value is empty.

icl_instance_path — An optional path specified in front of the iproc_name inside the iCall command, or in front of the icl_object inside other PDL commands such as iWrite, iRead, and iClock.

The effective_icl_instance_path is illustrated in the examples that follow.

The other arguments of this command are passed to the iProc in the same order as they appear in the command line. You must specify at least as many arguments as there are variables without default values in the corresponding iProc specification. For arguments defined with a default value in the associated iProc, their value takes on the default value when omitted.

Arguments

- [icl_instance_path.]**iproc_name**

  Name of an iProc preceded by an optional icl_instance_path and separated by a dot. The specified iProc must have been registered against the ICL module associated with the *effective_icl_instance_path*.

- arguments...

  Additional optional arguments that are passed to the iProc procedure. The number of required arguments depends on the definition of the iProc and on the existence of default values for the iProc arguments.

Return value

None. Unlike normal Tcl proc, even if the iProc contains the Tcl return command, the specified value is not returned by the iCall command which is consistent with the IJTAG 1687 standard.

Example

The following example assumes an ICL module called *myInstrument1* exists and this module is instantiated four times below the current design with the following instance paths:

core1_i1.myInst1_i1
core1_i1.myInst1_i2
core1_i2.myInst1_i1
core1_i2.myInst1_i2

The ICL module *myInstrument1* has a ICL sub-module called *myInstrument1_decode_logic* which is instantiated within *myInstrument1* as "*decode_logic*".

This numbered example illustrates the declarations of iProcs against the myInstrument1 and myInstrument_decode_logic modules, and the use of the iCall command to execute them against various instances of myInstrument1.

Line 1 issues the first iProcsForModule command for myInstrument1 which means iProcs declared afterward are registered against that module. For that reason, the iProcs declared at line 2 and 12 are registered against myInstrument1.

Line 24 issues the second iProcsForModule command which means the next iProcs are registered against the module myInstrument1_decode_logic.

The commands starting at line 41 are not inside an iProc so the iProc instance path value is empty when computing the effective_icl_instance_path for those commands.

The iCall on line 42 has an effective_icl_instance_path of core1_i1.myInst1_i1 because iPrefix path is currently set to core1_i1 (from the iPrefix command on line 41) and icl_instance_path is specified as myInst1_i1.

The iCall on line 42 specifies the value for the cnt argument while the second argument is left to the default value specified within the iProc declaration.

The iCall on line 43 overwrites the value of the second argument. Note that the first argument is required because it has no default values defined in the iProc declaration.

The iCall on line 5 is inside of an iProc, so it has an iProc instance path value that is equal to the effective_icl_instance_path value of the iCall that called the iProc. Looking at the iCall command on line 5 when called by the iCall on line 41, its effective_icl_instance_path is core1_i1.myInst1_i1.decode_logic because iProc instance path is core1_i1.myInst1_i1, iPrefix path is decode_logic, and icl_instance_path is empty.

The ICL instance core1_i1.myInst1_i1.decode_logic is an instance of ICL module myInstrument1_decode_logic and the proc Setup is registered against this module on line 25 because it follows the iProcsForModule command specified on line 24.

```
1  iProcsForModule myInstrument1
2  iProc run_testa {cnt {mode red}} {
3     iCall CheckStart
4     iPrefix decode_logic
5     iCall Setup $cnt $mode
6     iCall run $mode
7     iPrefix
8     iCall decode_logic.Status
10    iRead decode_logic.cnt $cnt
11 }
12 iProc CheckStart {} {
13    iRead go Fail
14    iRead done No
15    iApply
16 }
17 iProc run {mode} {
18    set cycles(red) 50
19    set cycles(blue) 90
20    set cycles(green) 70
21
22    iRunLoop $cycles($mode) -tck
23 }
24 iProcsForModule myInstrument1_decode_logic
25 iProc Setup { cnt mode } {
26    iWrite mode $mode
27    if {$mode == "red"} {
28       iWrite p1 0b0
29       iWrite p1(2) 0b1
30    }
31    iApply
32    iWrite enable Yes
33    iApply
34 }
35 iProc Status {} {
36    iRead done Yes
37    iRead go Pass
38    iRead count $cnt
39    iApply
40 }
41
41 iPrefix core1_i1
42 iCall myInst1_i1.run_testa 5
43 iCall myInst1_i2.run_testa 10 blue
44 iPrefix
45 iCall core1_i2.myInst1_i1.run_testa 15 red
```

Related Commands

iPrefix
iProc
iProcsForModule
iWrite

iClock

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iClock [icl_instance_path.]icl_clock_port

Description

Checks whether a controlling clock path from a valid clock source to the specified clock port exists.

This command takes into account the iClockOverride specifications and the FreqMultiplier and FreqDivider properties in the ICL description.

Prior to issuing this command, clock path sensitization must have been performed using the appropriate iWrite and iApply commands such that *SelectedBy* sources of ICL ClockMultiplexers seen along the traced path have known constant values.

If clock tracing fails before reaching a primary input, an error is generated. If the primary input is reached but does not have an associated add_clocks command to specify the clock is active with a known period, an error is also generated.

The cumulative effective frequency multiplier is computed along the traced clock path and the final number is stored against the specified clock port. This value is later used to normalize the iRunLoop command to a common tester period as described in the iRunLoop command description.

Arguments

- icl_instance_path.]icl_clock_port

  Name of an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

Return Value

None

Example

In the following example, an iClock command is first specified to check that the reference source pin of the PLL has a controlling source to a primary input of the current design, and that the port has a specified clock period as defined by an add_clocks command.

The iWrite and iApply commands then program the PLL to a multiplication factor of 1.5 and set the clock multiplexer to the desired state.

The result of the PLL setup is reflected by the iClockOverride command such that the iClock definition on inst1.mbist_i1.clk, which traces to the PLL, properly computes the correct multiplication factor.

The first iRunLoop command specifies that the PLL will lock in less that 200 reference clock cycles. The second iRunLoop results in a loop of 4000 (6000*2/3) clock cycles at the current design boundary. For more information on the iRunLoop scaling algorithm, see the iRunLoop command description.

```
iClock inst1.pll.ref
iWrite inst1.pll_setup.mult 0b0100
iWrite inst1.pll_setup.source 0b01
iApply
iClockOverride inst1.pll.vco -freqMultiplier 3 -freqDivider 2
iRunLoop 200 -sck inst1.pll.ref
iClock inst1.mbist_i1.clk
iRunLoop 6000 -sck inst1.mbist_i1.clk
```

Related Commands

add_clocks
delete_clocks
get_iclock_list
get_iclock_option
iClockOverride
iRunLoop
report_clocks
report_icl_modules

iClockOverride

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iClockOverride **[icl_instance_path.]icl_clock_port** -source [icl_instance_path.]icl_clock_source_port [-freqDivider divider] [-freqMultiplier multiplier]

iClockOverride **[icl_instance_path.]icl_clock_port** [-freqDivider divider] [-freqMultiplier multiplier]

Description

Models how the functional clocking has been programmed.

System clocking syntax inside ICL is adequate to describe a constant frequency multiplication and division number. With ClockMultiplexers in ICL, it is possible to model alternative sources as long as the select lines of the multiplexers are controlled by ijtag registers. The functional clocking path of ijtag instruments is not required to be controlled by ijtag itself. As long as the current functional clocking is properly modeled, the iRunLoop commands can be normalized correctly.

You should have a way to set up the functional clock using the ijtag network if you need to change it between patterns. You can use the complete syntax available inside the test_setup procedure to program an arbitrary functional clock system. You simply use this iClockOverride command to model how the functional clocking was programmed. Even if the multiplication factor is controlled by ijtag registers, you use the iClockOverride command to describe the effect of the newly written value. This is illustrated in the iClock command example.

You can use the iClockOverride command to override only the source, or only the cumulative multiplication/divider value, or both. If the -source option is not specified, the source is unchanged; it remains the source documented in ICL, or the last iClockOverride -source value that may have been specified within the current open pattern set.

If the -freqDivider and the-freqMultiplier arguments are both specified, both are overridden. Specifying just -freqDivider without -freqMultiplier or vice versa, sets the unspecified option to 1. If neither of the -freqDivider and the-freqMultiplier arguments are specified, the previous -freqDivider and the-freqMultiplier values are preserved, regardless of whether they came from ICL or a previously specified iClockOverride.

Arguments

- **[icl_instance_path.]icl_clock_port**

  A required value that specifies an internal clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- -source [icl_instance_path.]icl_clock_source_port

  An optional switch and value pair that specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The iCall command description describes how the effective_icl_instance_path value is computed and the factors that affect it.

- -freqDivider *divider*

  An optional option-value pair that overrides the frequency ratio between the *[icl_instance_path.]**icl_clock_port*** and its source. It's value defaults to 1 when -freqMultiplier is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]**icl_clock_port***

- -freqMultiplier *divider*

  An optional option-value pair that overrides the frequency multiplication factor between the *[icl_instance_path.]**icl_clock_port*** and its source. It's value defaults to 1 when -freqDivider is specified otherwise it defaults to the previously defined value, either from the definition in ICL or from a previous iClockOverride command on the given *[icl_instance_path.]**icl_clock_port***

Return value

None

Example

The following example assumes both inst1.inst2.sysclock and clk2 are declared as ClockPorts in the ICL description. The first iClockOveride command only overrides the source and leaves the multiplication and division number unchanged. The second iClockOveride command overrides the frequency multiplication factor to 10 and the frequency division ratio to 1.

```
iClockOverride inst1.inst2.sysclock -source clk2
iClockOverride inst1.inst2.sysclock -freqMultiplication 10
```

Related Commands

add_clocks
delete_clocks
get_iclock_list
get_iclock_option
iClock
iRunLoop
report_clocks
report_icl_modules

identify_redundant_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

identify_redundant_faults

Description

Identifies redundant faults among current faults whose redundancy has not yet been determined.

Note

By default, the create_patterns command automatically performs redundant fault identification at the conclusion of the run.

Identifies combinationally redundant (RE) faults from among those faults in the current fault list whose redundancy has not been determined yet.

This command is useful in situations where the tool has not specifically checked some faults for redundancy, but where establishing their redundancy might improve test coverage. The following are examples of such situations:

- When the tool generates patterns using named capture procedures, it does not identify RE faults. You might improve test coverage by using this command at the end of such an ATPG session.
- When the tool simulates external patterns, it does not identify RE faults among the undetected faults. You might improve test coverage by using this command after the simulation.

This command will only target faults not yet proven to be either RE or non-RE. For example, the command will not target a fault currently classified as det_simulation (DS) or that was classified as DS before but is now uncontrolled (UC) or unobserved (UO). The command is meaningful only when the current fault type is stuck-at, bridge, transition, or UDFM.

Example 1

The following example creates patterns using named capture procedures, then identifies the RE faults:

```
set_system_mode analysis
set_capture_procedures on -all
create_patterns

...
//  There may be unidentified redundant faults when generating patterns
//     with named capture procedures defined, which may lead to lower test
//     coverage being reported.
//  To perform redundant fault identification, invoke the command:
//     identify_redundant_faults
...
```

```
            Statistics report
-------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                 471724    783223
-----------------------   -------   -------
UC (uncontrolled)             39       173
UO (unobserved)             2748      4490
DS (det_simulation)       380299    652427
DI (det_implication)       66661     88877
PU (posdet_untestable)        26        36
PT (posdet_testable)         147       223
BL (blocked)                1907      3010
AU (atpg_untestable)       19897     33987
-----------------------   -------   -------
test_coverage              95.17%    95.05%
fault_coverage             94.79%    94.68%
atpg_effectiveness         99.41%    99.40%
-------------------------------------------
#test_patterns                         2736
    #clock_sequential_patterns         2736
#simulated_patterns                    2880
CPU_time (secs)                       793.1
-------------------------------------------
```

identify_redundant_faults

```
//  deterministic ATPG invoked with comb/seq abort limit = 0/2
//  ----------------------------------------------------------------------
//  Performing redundant fault identification for 22857 faults
//  ----------------------------------------------------------------------
//  # red.   # non-red.  # abort   # remn.   progress  test      process
//  faults     faults     faults    faults             coverage  CPU time
//    261       12977        97       9619    57.92%   95.22%     3.26 sec
//   1361       16215       421       5281    76.90%   95.45%     4.80 sec
//   2846       18269       849       1742    92.38%   95.75%     6.67 sec
//   3121       18896       922          0   100.00%   95.81%     6.89 sec
//
```

report_statistics

```
            Statistics report
-------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                 471724    783223
-----------------------   -------   -------
UC (uncontrolled)             39       173
UO (unobserved)             2290      3832
DS (det_simulation)       380299    652427
DI (det_implication)       66661     88877
PU (posdet_untestable)        26        36
PT (posdet_testable)         145       221
BL (blocked)                1907      3010
RE (redundant)              3121      4322
AU (atpg_untestable)       17236     30325
```

```
-----------------------   -------   -------
test_coverage              95.81%    95.58%
fault_coverage             94.79%    94.68%
atpg_effectiveness         99.51%    99.49%
-----------------------------------------
#test_patterns                        2736
    #clock_sequential_patterns        2736
#simulated_patterns                   2880
CPU_time (secs)                      802.8
-----------------------------------------
```

Example 2

The following example shows the command flow to identify the redundant faults after simulating external patterns that were created using named capture procedures:

set_system_mode analysis
read_patterns my_external_patt
simulate_patterns
identify_redundant_faults

Related Commands

create_patterns
report_statistics
simulate_patterns
set_capture_procedures
set_fault_type

iMerge

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iMerge {-begin [-error_on_conflict] | -end}

Description

Specifies the beginning and the end of a set of iCall commands that are meant to be processed in parallel, as much as possible, in the final representation of the test patterns.

iCall is the only PDL command which is allowed in between iMerge -begin and iMerge -end.

In the iProcs called by the iCalls, Tessent IJTAG identifies any resource conflicts, such as two PDL commands writing to the same register, and all conflicts arising from reservations done by means of the command iTake. All conflicting commands are then processed serially. For the remaining iCall commands, Tessent IJTAG tries to find an optimal parallel solution.

Arguments

- **-begin**

  A required switch that marks the beginning of the set of iCall commands to be processed in parallel.

- **-error_on_conflict**

  An optional switch that specifies to issue an error when there is a conflict rather than process the conflicting events serially. In the case of nested iMerge blocks, you only need to specify this switch on the outermost iMerge specification; the specification applies to all iMerge processes that are called from within the block.

  Refer to “iMerge Conflict Reporting” in the *Tessent IJTAG User’s Manual* for more information.

- **-end**

  A required switch that marks the end of the set of iCall commands to be processed in parallel.

Return value

None.

Example

```
iMerge -begin -error_on_conflict
iCall proc1
iCall proc2
iCall proc3
…
iMerge -end
```

Related Commands

iCall
iPrefix
iTake

import_dfm

Context: patterns -scan_diagnosis

Mode: analysis

Usage

import_dfm **-rdb** ***rdb_file*** **-layer** ***layer_name*** **-type** {**cell** | **open** | **bridge**}
[-sample *n*] [-replace]

Description

Imports a Calibre-compatible results database (RDB) file into a violations database (VDB).

The import_dfm command verifies that the DFM rule violations in the VDB match the layout geometries in the LDB. It then populates the VDB with the DFM rule violations.

Note

DFM shapes that fail verification are still imported into the VDB and used during annotation.

You must have previously opened the LDB and VDB with the open_layout command. Otherwise, Tessent Diagnosis issues an error message.

For more information, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- **-rdb** ***rdb_file***

  A required switch and string pair that specifies the Calibre-compatible DFM RDB file.

- **-layer** ***layer_name***

  A required switch and string pair that specifies the layer. You must specify a known layer in the layout-aware diagnosis layout file.

- **-type** {**cell** | **open** | **bridge**}

  A required switch and literal that specifies the type of DFM violation that you want to report.

- -sample *n*

  An optional switch and string pair that specifies a percentage value from 0 to 100. This value specifies the number of shapes that will be verified against the LDB. Setting this value to 0 prohibits verification.

  This switch has no effect on diagnosis. Use it to improve verification runtimes with the caveat that lowering the value decreases the verification accuracy.

- -replace

  An optional switch that specifies to remove from the VDB the DFM rule with the same rule name, and layer, and to replace it with the new RDB information.

Examples

The following example opens an LDB, creates the directory into which Tessent Diagnosis will write the VDB, and then imports the RDB files into the VDB created in the design.lddb directory. During the verification step, it verifies 10% of the rule violations in the VDB.

```
open_layout design.ldb design.lddb
import_dfm -rdb id_100.rdb -layer route_2 -type OPEN -sample 10 -replace
```

Related Commands

annotate_diagnosis

delete_dfm

index_collection

Context: unspecified, all contexts

Mode: all modes

Usage

index_collection ***collection index***

Description

Extracts a single object at a specified index in a collection and creates a new collection containing only that object.

The base collection remains unchanged.

Arguments

- ***collection***

  A required string that specifies the collection to be searched.

- ***index***

  A required integer that specifies the index into the collection. Allowed values are from 0 to *sizeof_collection* - 1.

Example

The following example shows the first and last object in a collection being extracted.

```
set c1 [get_instances {u1 u2 u3}]
{u1 u2 u3}

get_name_list [index_collection $c1 0]
u1

get_name_list [index_collection $c1 [expr [sizeof_collection $c1] -1 ]]
u3
```

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
is_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

iNote

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iNote **string**

Description

Inserts a note in the opened pattern set.

The note is attached to the time corresponding to where the iNote command is executed. When the pattern set is written to a pattern file by means of write_patterns commands, the note will appear as an annotation in the pattern file at the appropriate location. For simulation test benches, the note is displayed when the corresponding cycle to which the note is attached to is simulated.

Arguments

- **string**

  The comment string or note that will be added to the pattern set at the proper time. This must be a single normal Tcl string, Multiple words must be enclosed in quotes or braces.

Return value

None

Example

In the following example, the first iNote command is attached to the beginning of the pattern. The iNote string contains a variable that will be substituted before being sent to the iNote command. The second iNote command is attached to the time corresponding to the time reached to execute the previous iApply actions. The last iNote appears after the loop.

```
iNote "Setting up controller instance $inst"
iWrite mode blue; iApply
iNote “Running controller”
iWrite enable On; iApply
iRunLoop 30 -tck
iNote “Checking results”
iRead results pass; iApply
```

Related Commands

None

insert_test_logic

Context: dft -scan, dft -test_points

Mode: analysis

Prerequisites: To use the *-Verilog* switch, you must have defined a buffer model with the add_cell_models command or with a cell_type attribute.

Usage

Context: dft -test_points

insert_test_logic

Context: dft -scan

insert_test_logic [*filename* [-Fixed]] [-Scan {ON | OFf}] [-Test_point {ON | OFf}]
[-Ram {ON | OFf}] {[-UNlimited] | [-MAx_length *integer*] | [-NUmber [*integer*]]}
[-Clock {Nomerge | Merge}] [-Edge {Nomerge | Merge}]
[-COnnect {ON | OFf | Tied | Loop | Buffer}] [-Output {Share | New}]
[-MOdule {Norename | Rename}] [-Verilog] [-NEw_scan_po] [-Keep_original_net]

Description

Inserts the test structures you define into the netlist to increase the design's testability.

The tool inserts test structures into a design to increase the fault coverage (testability) of the design. Specifically, the tool attempts to increase a design's fault coverage with the insert_test_logic command by doing the following:

- Identifies sequential cells and replaces them with corresponding scan cells, and stitches them together into a scan chain. (Referred to as scan cell replacement and stitching.)
- Supports the adding of both system-defined and user-defined test points.
- Supports the automatic adding of test logic.

If you issue the insert_test_logic command after the set_wrapper_chains command, but have not issued the analyze_wrapper_cells command, you will get an error messsage stating that the wrapper chains analysis is required prior to insertion.

```
Error: Wrapper cells information is out of date:use
'analyze_wrapper_cells' to re-analyze wrapper cells prior to insertion.
```

For more information on scan replacement, test points, and test logic, refer to the *Tessent Tessent Scan and ATPG User's Manual*. For information on test point insertion, refer to "Manually Specifying Control and Observe Points" in the *Tessent Scan and ATPG User's Manual*. For information on handling pre-inserted scan cells, refer to "Specifying Existing Scan Cells" in the *Tessent Scan and ATPG User's Manual*.

Note

You can specify either a number chain specification or a maximum length chain specification in a single session among the insert_test_logic, add_scan_partition, and set_wrapper_chains commands, but, you cannot specify both. For example, if you specify the number of chains for input wrapper chains using the set_wrapper_chains -input_number option, you cannot use the insert_test_logic -max_length option for the remainder of the scan cells.

Arguments

- *filename* -Fixed

  An optional string and associated optional switch that specify the name of the ASCII file that lists the scan instances that you want the tool to stitch together. This file can contain information regarding scan cell ordering along with which instances are to be in each scan chain.

Note

If you use this file, it must contain all instances you want stitched. If you do not specify a filename, the tool stitches all non-scan cells that it has identified and mapped to scan cells into a scan chain using the settings of the other insert_test_logic arguments.

The -Fixed switch specifies for the tool to stitch the scan instances in the fixed order that is given in the filename. The default scan cell ordering is based on hierarchical rules, but within a hierarchical block the scan cell ordering is random. When using the -Fixed option, it also ignores certain scan input/output mapping performed by the add_scan_pins command.

For more information, refer to "Naming Scan Input and Output Ports" in the *Tessent Scan and ATPG User's Manual*.

The *filename* that you specify must list one instance per line and use the following format (all on one line):

```
instance_pathname cell_id chain_id [&sub_chain_name]
   [{+|-}[lockup_latch_model]]
```

instance_pathname — A string that specifies the name of the non-scan cell that you want the tool to put in the scan chain.

cell_id — An integer that specifies the placement of the instance_pathname in relation to other instance_pathnames. The tool places the instance having the smallest cell_id closest to the scan chain output. All instances in the same chain must have unique cell_ids.

chain_id — An integer that specifies the scan chain in which you want the tool to place the instance_pathname. The tool places instances with the same chain_id in the same chain.

&sub_chain_name — An optional special character and string that specifies the name of the sub-scan chain you defined with the add_sub_chains command. You need to

specify the subchain name when more than one subchain is defined for a sub-module or a hierarchical instance. No space is allowed between the ampersand (&) and the sub_chain_name argument.

{+|-}[*lockup_latch_model*] — An optional special character and an additional optional string that specifies the location of the lockup cell. The +|- argument specifies that a lockup cell is to be added to the scan out of the current instance_pathname. No space is allowed between the +|- and the lockup_latch_model name. Note that if you define a lockup cell in this file, you must specify every location that you want to insert lockup cells. If no lockup cells are defined in this file, the tool uses the settings in the set_lockup_cell and insert_test_logic commands. In such case, if you want the tool to automatically insert lockup cells in necessary locations, you should use the "-Clock/-Edge merge" option(s) with the insert_test_logic command. For more information on inserting lockup cells, refer to the set_lockup_cell command and "Merging Scan Chains with Different Shift Clocks" in the *Tessent Scan and ATPG User's Manual*.

+ — Specifies to use the clock used by the instance_pathname with the next lower cell_id. The next lower cell_id, refers to the a non-scan cell which will be connected to the scan out of the current *instance_pathname*.

- — Specifies to use the clock used by the *instance_pathname*.

lockup_latch_model — Specifies the name of the lockup cell model to use. The specified lockup cell must be defined by the add_cell_models command or defined in the ATPG library using the cell_type attribute. If the lockup_latch_model is not specified, the first model in the defined latch model list is used.

- -Scan ON | OFf

  An optional switch and literal pair that specifies whether the tool replaces the identified non-scan cells (scan candidates) with the corresponding scan cells. The valid literals are as follows:

  ON — A literal that enables the tool to replace the identified non-scan cells (scan candidates) with the corresponding scan cells. This is the default.

  OFf — A literal that disables the tool from replacing the identified non-scan cells (scan candidates) with the corresponding scan cells.

- -Test_point ON | OFf

  An optional switch and literal pair that specifies whether the tool adds the identified test logic and test points into the design. The valid literals are as follows:

  ON — A literal that enables the tool to add the identified test logic and test points into the design. This is the default.

  OFf — A literal that disables the tool from adding the identified test logic and test points into the design.

- -Ram ON | OFf

  An optional switch and literal pair that specifies whether the tool adds the identified test logic gates that are necessary to allow the ATPG tools access to the write control lines of the RAMs. The valid literals are as follows:

ON — A literal that enables the tool to add the identified test logic gates for RAM write control line access. This is the default.

OFf — A literal that disables the tool from adding the identified test logic gates for RAM write control line access.

- -UNlimited

  An optional switch specifying that the scan chain has no limit on the number of scan cells it contains. This is the default.

- -MAx_length *integer*

  An optional switch and integer pair that specifies the maximum number of scan cells that the tool can stitch into a scan chain. The tool evenly divides the scan cells into scan chains that are smaller than the max_length integer. Final results depend upon the number of scan candidates.

- -NUmber *integer*

  An optional switch and integer pair that specifies the exact number of scan chains that you want the tool to insert. Final results depend upon the number of scan candidates. The default number of chains is 1.

Note

The number of scan chains specified with insert_test_logic *-number* applies to the entire design. The number of scan chains specified with other commands such as set_wrapper_chains and add_scan_partition is subtracted from this number.

- -CLock Nomerge | Merge

  An optional switch and literal pair that specifies whether the tool uses different clocks on the same scan chain. The two valid literals are as follows:

  Nomerge — A literal that disables the use of different clocks on the same scan chain. This is the default.

  Merge — A literal that enables the use of different clocks on the same scan chain.

- -Edge Nomerge | Merge

  An optional switch and literal pair that specifies whether the tool merges stable high chains into stable low chains. The two valid literals are as follows:

  Nomerge — A literal that specifies to not merge stable high chains into stable low chains. This is the default.

  Merge — A literal that specifies to merge stable high chains into stable low chains.

- -COnnect ON | OFf | Tied | Loop | Buffer

  An optional switch and literal pair that specifies whether the tool stitches the scan cells together into a scan chain. The valid literals for stitching the scan chain are as follows:

ON — A literal that specifies for the tool to replace the identified non-scan cells with their corresponding scan replacements and to stitch those scan cells together into a scan chain. This is the default.

OFf — A literal that specifies for the tool to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains.

Tied — A literal that specifies for the tool to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This option has the tool tie the input/output scan pins to ground.

Loop — A literal that specifies for the tool to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This option has the tool connect the scan_out pin to its own scan_in pin as a self-loop.

Buffer — A literal that specifies for the tool to replace the identified non-scan cells with their corresponding scan replacements, but not stitch those scan cells together into scan chains. This option has the tool connect the scan_out pin to its own scan_in pin as a self-loop with a buffer in between.

- -Output Share | New

  An optional switch and literal pair that specifies how the tool creates scan out ports on modules. The valid literals are as follows:

  Share — A literal specifying that the tool may use an existing module output port on modules for scan out, if that port is directly connected to the scan out of a scan cell. This is the default.

  New — A literal specifying that the tool should always create a new output port for scan out.

Note

If you want the tool to only create new scan output ports on the top-level module, use the *-NEw_scan_po* switch, instead of the *-Output* switch.

- -MOdule Norename | Rename

  An optional switch and literal pair that specifies how to name the modified module. The valid literals are as follows:

  Norename — A literal specifying that the tool should use the original module name, if it uses only one type of module modification, and that the original module is no longer used in the design. This is the default.

  Rename — A literal specifying that the tool should always rename a module if it modifies the module.

- -Verilog

  An optional switch that specifies to the tool that the final output netlist format will be Verilog, so that the tool knows to insert a buffer instance for any scan output pin in any module in the design hierarchy, if that pin also fans out as the module's functional output.

Without this switch, the tool uses the Verilog 'assign' statement to generate this scan output signal from the functional output. However, some layout tools do not support the Verilog assign statement, thus the -Verilog switch is required so that the tool will use a buffer instantiation and avoid using the assign statement.

Before using this command and switch, you must have defined a buffer model with the add_cell_models command or with a cell_type attribute.

- -NEw_scan_po

  An optional switch that specifies for the tool to create new scan primary output pins even though existing functional outputs are available to use as scan outputs. The switch is a special case of the -Output switch used with "New" literal, as it performs new scan output port creation only on the top-level module, instead of all modules.

- -Keep_original_net

  An optional switch that specifies for the tool to insert a buffer in the scan path, between the last scan cell at the top-level and the top-level scan output pin, when the scan cell output has no connection to top-level but has a functional connection to other logic. The buffer insertion is done to prevent renaming the original net that the scan cell is connected to the other logic with. When this switch is not specified, the tool renames the original net same as the primary output pin that it creates as scan output port.

Examples

The following example identifies the scannable sequential instances and then stitches them into scan chains when the insert_test_logic command executes:

```
add_clocks 0 clock
set_system_mode analysis
insert_test_logic -scan on -max_length 10
```

Related Commands

add_scan_instances
add_tied_signals
analyze_wrapper_cells
report_scan_chains
set_test_logic
set_scan_insertion
set_test_point_insertion_options
set_wrapper_chains

intercept_connection

Context: dft

Mode: insertion

Usage

intercept_connection **node -cell_function_name *function_name***
[-technology *dft_cell_selection_name*]
[-input2 *input2_source*] [-select *select_source*]
[-leaf_instance_prefix *prefix_name*]
[-only_when_has_functional_source]

Description

A command that uses the get_dft_cell command to obtain a cell with the specified function name and uses it to intercept a connection to a pin, port, or net.

This command performs interception inside both uniquified and not uniquified modules.

Arguments

- **node**

  A required value that is either a port, pin, or net name in the design or a collection of one port, pin, or net object. Based on the type of object and its direction attribute, in the case of a pin or a port, the source or destination connection of the object is intercepted. Table 7-17 shows the interception that is performed based on the node type. Note that an error is generated if the node is an inout pin or port, or if the node is a net corresponding to an inout pin or port.

Table 7-17. Interception based on node type

Node Type	Interception Done	Illustration
Input port	The node is a driver. The intercepting cell ends up between the input port and its original fanout.	

Table 7-17. Interception based on node type

Node Type	Interception Done	Illustration
Output port	The node is a sink. The intercepting cell ends up between the original fanin and the output port	Before and after interception:
Input pin	The node is a sink. The intercepting cell ends up between the original fanin and the input pin	Before and after interception:
Output pin	The node is a driver. The intercepting cell ends up between the output pin and its original fanout.	Before and after interception:

Table 7-17. Interception based on node type

Node Type	Interception Done	Illustration
Net corresponding to an input pin or an input port	The node is a driver. The intercepting cell ends up between the input pin or port and its original fanout.	Before and after interception:
Net corresponding to an output pin or an output port	The node is a sink. The intercepting cell ends up between the original fanin and the output pin or port.	Before and after interception:
Net not corresponding to a pin or a port	The node is a driver. The intercepting cell ends up between the driver of the net and its original fanout. Only single driver nets can be intercepted.	Before and after interception:

- **-cell_function_name *function_name***

  A required switch and string pair that specifies the function of the cell to be used for the interception. Table 7-18 shows the allowed cell function names and how the cell function affects the need to specify the -input2 and -select options.

Table 7-18. Allowed Cell Function Names

Function Name	Results
and	The interception is performed with the 2 input AND cell returned by the "get_dft_cell and" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see AND Cells for more details about the "and" cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option.
buffer	The interception is performed with the BUFFER cell returned by the "get_dft_cell buffer" command. In the *Tessent Cell Library Manual*, see Buffer and Input Pad for more details about the "buffer" cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option.
inverter	The interception is performed with the INVERTER cell returned by the "get_dft_cell inverter" command. In the *Tessent Cell Library Manual*, see Inverter for more details about the "inverter" cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option.
mux	The interception is performed with the MUX cell returned by the "get_dft_cell mux" command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the *Tessent Cell Library Manual*, see MUX Cell for more details about the "mux" cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
or	The interception is performed with the 2 input OR cell returned by the "get_dft_cell or" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see OR Gate for more details about the "or" cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option.
clock_and	The interception is performed with the 2 input clock AND cell returned by the "get_dft_cell clock_and" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see AND Cells for more details about the "clock_and" cell_type and Cell Selection to know how to control which AND cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function clock_in, it is selected for the interception.

Table 7-18. Allowed Cell Function Names

Function Name	Results
clock_buffer	The interception is performed with the clock BUFFER cell returned by the "get_dft_cell clock_buffer" command. In the *Tessent Cell Library Manual*, see Buffer and Input Pad for more details about the "clock_buffer" cell_type and Cell Selection to know how to control which BUFFER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.
clock_inverter	The interception is performed with the clock INVERTER cell returned by the "get_dft_cell clock_inverter" command. In the *Tessent Cell Library Manual*, see Inverter for more details about the "clock_inverter" cell_type and Cell Selection to know how to control which INVERTER cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library.
clock_mux	The interception is performed with the MUX cell returned by the "get_dft_cell clock_mux" command. The -input2 and -select options are required to specify the source of the second data input and the select input. In the *Tessent Cell Library Manual*, see MUX Cell for more details about the "clock_mux" cell_type and Cell Selection to know how to control which MUX cell is used based on the specified -technology option.
clock_or	The interception is performed with the 2 input clock OR cell returned by the "get_dft_cell clock_or" command. The -input2 option is required to specify the source of the second input. In the *Tessent Cell Library Manual*, see OR Gate for more details about the "clock_or" cell_type and Cell Selection to know how to control which OR cell is used based on the specified -technology option. The clock type is not learned and must be explicitly specified in the library. If one of the inputs has the function clock_in, it is selected for the interception.

- -technology *dft_cell_selection_name*

  An optional switch and value pair that specifies from which dft_cell_selection list the cell is to be used. You only need to specify this option if your library has more than one dft_cell_selection wrapper. If you have none, one is learned for the simple cell types, but you will not have access to the clock cells unless you have set their cell_type attribute in the library. See "Model-Level Attribute Descriptions" in the *Tessent Cell Library Manual* for information on how to specify the cell_type attribute.

- -input2 *input2_source*

  An optional switch that specifies a node to connect to the second input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins

at some later time. Table 7-18 describes which values for -cell_function_name makes the specification of this option mandatory.

- -select *select_source*

  An optional switch and value pair that specifies a node to connect to the select input of the cell. The source must be a valid pin, port, or net object, or a name, or the constant value 0 or 1. You should use a constant value if you want to insert a gate now and then connect the extra pins at some later time. Table 7-18 describes which values for -cell_function_name make the specification of the -select option mandatory.

- -leaf_instance_prefix *prefix_name*

  This optional switch and value pair specifies the prefix to used when instantiating the cell. When this is not specified, the default is “insertion_”. The leaf instance name is constructed by concatenating the prefix_name with the cell_function_name. When a conflict is detected during instantiation, the tool automatically adds a “_<int>” suffix to uniquify the leaf name. You can use a specific prefix if you want to easily find the inserted cells in a synthesis or layout script by using a command such as “get_cells *prefix_name* -hier”.

- -only_when_has_functional_source

  This optional switch specifies to skip the instantiation of the cell and interception of the node when the attribute has_functional_source is 0 for the node. Instead, the node is disconnected and driven by the specified -input2 value. You cannot specify this option when -cell_function_name refers to a single input cell as there is no -input2 value to connect to in that case.

Example 1

The following example inserts a simple inverter gate by intercepting the existing connection between the u7/D pin and the u6/Y pin, shown on the left side of Figure 7-16. After the interception, the inverter A pin is connected to the u6/Y pin and the inverter Y pin is connected to the u7/D pin as shown on the right side of Figure 7-16.

INSERTION> intercept_connection u7/D -cell_function_name inverter

```
{insertion_inverter}
```

Figure 7-16. Inverter Interception

Example 2

The following example inserts a multiplexer and specifies to connect the second multiplexer input pin to 1, and to connect the multiplexer select pin to scan_en.

INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 -select scan_en

```
{u4/insertion_mux_1}
```

This example does the same thing as first example above but specifies the -only_when_has_functional_source option. Because this option is specified, the tool connects the pin u1/u3/D directly to a constant 1 if its has_functional_source attribute evaluates to 0. In this case, the commands returns null as no intercepting cell was instantiated.

INSERTION> intercept_connection u4/u3/D -cell_function_name mux -input2 1 -select scan_en -only_when_has_functional_source

```
{}
```

Example 3

The following example script shows how to cascade three gates on a set of nodes. The example illustrates how the command is capable of handling non-unique designs. The script will work whether module “rep3” is uniquified or not. Notice how the example inserts the first gate level on all nets, and then adds two more levels in a row on each net. This illustrates that it does not matter if you insert all gate levels on each net or if you add one level at a time. In both cases, the result is identical.

Original design:

```
module ex (ina, outa, c1, c2, c3);

  input c1, c2, c3;
  input [3:0] ina;
  output [3:0] outa;

  rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]));
  rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]));
  rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]));

  rep1 rep1 (.ina(ina[3]), .outa(outa[3]));

endmodule

module rep3 (ina, outa);
  input ina;
  output outa;

  assign outa = ina;

endmodule

module rep1 (ina, outa);
  input ina;
  output outa;
```

```
  assign outa = ina;

endmodule
```

The dofile:

```
set nets [get_nets -of_pins [get_pins -direction input]]
foreach_in_collection net $nets {
  intercept_connection $net -cell or -input2 c1}

foreach_in_collection net $nets {
  foreach var{2 3} {
   intercept_connection $net -cell [lindex {and or} [expr $l-2]] \
            -input2 c$var -leaf_inst_prefix myprefix_
  }
}
```

The modified design:

```
module ex (ina, outa, c1, c2, c3);

  input c1, c2, c3;
  input [3:0] ina;
  output [3:0] outa;

  rep3 rep3_0 (.ina(ina[0]), .outa(outa[0]), .c1(c1), .c2(c2), .c3(c3));
  rep3 rep3_1 (.ina(ina[1]), .outa(outa[1]), .c1(c1), .c2(c2), .c3(c3));
  rep3 rep3_2 (.ina(ina[2]), .outa(outa[2]), .c1(c1), .c2(c2), .c3(c3));

  rep1 rep1 (.ina(ina[3]), .outa(outa[3]), .c1(c1), .c2(c2), .c3(c3));

endmodule

module rep3 (ina, outa, c1, c2, c3);
  input ina;
  output outa;
  input c1, c2, c3;
  wire c1, c2, c3;

  wire ina_ts1, ina_ts2;

  OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));

  AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));

  OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule

module rep1 (ina, outa, c1, c2, c3);
  input ina;
  output outa;
  input c1, c2, c3;
  wire c1, c2, c3;

  wire ina_ts1, ina_ts2;

  OR210 insertion_or(.A(ina_ts1), .B(c1), .Y(outa));
```

```
  AN210 myprefix_and(.A(ina_ts2), .B(c2), .Y(ina_ts1));

  OR210 myprefix_or(.A(ina), .B(c3), .Y(ina_ts2));
endmodule
```

Related Commands

get_dft_cell

move_connections

iPrefix

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iPrefix [icl_instance_path]

Description

Sets the iPrefix path that is used to compute the effective_icl_instance_path for the iCall command and other PDL commands.

The effective_icl_instance_path is described in detail in the iCall command description.

This command is not affected by the previous iPrefix command. If the *icl_instance_path* argument is omitted, the iPrefix path value is set to empty. If the path is specified and the iPrefix command is *inside* an iProc, it must be a valid instance path relative to the current iProc instance path. If the path is specified and the iPrefix command is *not* inside an iProc, it must be a valid instance path relative to the current design.

Arguments

- icl_instance_path

  An optional string that defines the new value for iPrefix path used to compute the effective_icl_instance_path for the iCall command and other PDL commands such as the iRead, iWrite, and iClock commands.

Return Value

None

Example

In the following example, the iPrefix path value is first set to block1_i1.inst1_i1. It is then set to block1_i2 with the second invocation of the iPrefix command. Notice how the second iPrefix command completely replaces the value of the iPrefix path and is not influenced by the previous value of iPrefix path. Only the iCall, iClock, iRead, iWrite, iRunLoop, and iClockOverride commands are affected by the iPrefix command.

```
iPrefix block1_i1.inst1_i1
iWrite A 0b0
iPrefix block1_i2
iWrite inst1_i1.A 0b0
```

Related Commands

iCall
iClock
iClockOverride
iRead
iRunLoop
iWrite

iProc

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The iProcsForModule has been specified.

Usage

iProc **name variables body**

Description

Specifies a PDL procedure that can run later when referenced by the iCall command.

Analogously to the Tcl command "proc", the specification consists of the procedure name, the procedure variables, and the procedure body.

The specified procedure is registered against a specific ICL module that must have been previously loaded by read_icl and specified by iProcsForModule. Since the iProc is registered against a certain module, several iProcs with the same name may exist, each associated to a different ICL module. If an iProc is specified with a name that already exists for the current iProcsForModule, the new specification replaces the existing one. If an ICL module is deleted, the related iProcs are deleted too.

When the procedure is called using the iCall command, the variables in the argument list are substituted with the values specified in the iCall command. The commands inside the iProc body are executed in the scope of the instance against which the iProc is called.

Arguments

- **name**

  Name of the iProc which must be a simple scalar identifier. A simple scalar identifier starts with a letter and contains only letters, decimal digits, and underscores.

- **variables**

  A Tcl list of variable name or variable-value pairs enclosed in curly braces. The list of variables can contain either simple identifiers or identifier-value pairs, where the value specifies the default value of the variable. All simple identifiers must be in the list before the identifier-value pairs.

- **body**

  Commands to be executed when the procedure is called by means of the iCall command.

Return Value

None

Example

The following example shows an iProc called run_testa that is registered against the myInstrument1 ICL module. The iProc has two arguments: cnt and mode. The mode argument

is optional because it has a specified default value of “red”. The body of the iProc contains standard Tcl syntax and PDL commands. The cnt argument is specified before the mode argument because it has no specified default value.

```
iProcsForModule myInstrument1

iProc run_testa {cnt {mode red}} {
  set cycles(red) 50
  set cycles(blue) 90
  set cycles(green) 70

  iRead go Fail
  iRead done No
  iApply

  iWrite mode $mode
  if {$mode == "red"} {
    iWrite p1 0b0
    iWrite p1(2) 0b1
  }
  iApply

  iWrite enable Yes
  iApply

  iRunLoop $cycles($mode) -tck

  iRead done Yes
  iRead go Pass
  iRead count $cnt
  iApply
}
```

Related Commands

delete_iproc
get_iproc_argument_default
get_iproc_argument_list
get_iproc_body
get_iproc_list
iCall
iProcsForModule

iProcsForModule

Context: all contexts

Mode: setup, analysis (patterns contexts only), insertion

Prerequisites: The corresponding ICL module already loaded.

Usage

iProcsForModule **icl_module_name [-iProcNameSpace** *name_space*]

Description

Specifies the ICL module for which subsequent iProc commands refer to.

An iProc is a PDL procedure written for a specific ICL module.

All references to ICL objects within the iProc refer to the instance path relative to the module it is written for. Once you have issued the iProcsForModule command, all iProcs defined afterward are registered against the specified ICL module. You cannot define any iProc until the iProcsForModule is set.

The iProcsForModule is specified once for each ICL module that you have iProcs to register against it. The iProcsForModule can only be specified for ICL modules that already have been loaded using the read_icl command.

To be sure that you load iProcs against the correct module, you can store all iProcs associated to a given ICL module inside a single file and add the proper iProcsForModule at the top of the file. With this organization, you can load the iProcs for any module as soon as the specified ICL module has been read in using the read_icl command.

Arguments

- **icl_module_name**

  Name of a valid ICL module which has already been loaded into memory using the read_icl command.

- **-iProcNameSpace** *name_space*

  An optional switch and string that creates a sub name space within each name space created from the ICL module name. When *name_space* is not specified, it defaults to the value set by the iUseProcNameSpace command.

Return Value

None

Example

In the following example, the file myCore.icl is read in; the file contains the entire ICL description of myCore. A child module of myCore is myInstrument1. The file *myInstrument1.iprocs* contains an iProcsForModule command that specifies that myProc defined afterward is meant to be registered against this module.

Sourcing the file myInstrument1.iprocs loads the iProcs defined in it. One of the instances of myInstrument1 within myCore is u2.u3. The iCall u2.u3.myProc command executes the iProc called *myProc* defined for module *myInstrument1* against the *u2.u3* ICL instance of that module. The solver will retarget the PDL found within that iProc to the boundary of myCore.

```
read_icl myCore.icl
source myInstrument1.iprocs

set_current_design myCore

set_system_mode analysis
open_pattern_set P1
iCall u2.u3.myProc
close_pattern_set
```

myInstrument1.iprocs file contains

```
iProcsForModule myInstrument1
iProc myProc {} {
   iRead u1.R 0b0
   iApply
}
```

Related Commands

delete_iproc
get_iproc_argument_default
get_iproc_argument_list
get_iproc_body
get_iproc_list
iCall
iProc

iRead

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iRead **[icl_instance_path.]readable_icl_object** [value]

Description

Adds a read operation to the command queue that will be solved by the next iApply command.

You can specify an optional expected value to which the read value will be compared to when observed. The iApply command description describes how the iRead commands are interpreted by the iApply command.

Arguments

- **[icl_instance_path.]readable_icl_object**

  Specifies the ICL objects that are to be read. You can specify to add read operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description.

- value

  An optional constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the readable_icl_object in the ICL description.

Return Value

None

Example

The following example requests the readable_icl_objects *done* and *go* to be read and compared against the value represented by the enumeration symbol called *Yes* and *Pass* defined in ICL. The ICL object *count* is scheduled to be read but not compared to any expected value. The three requested read operations are solved when the iApply command executes.

```
iRead done Yes
iRead go Pass
iRead count
iRead status 0b1
iApply
```

Related Commands

iApply
iCall
iPrefix
iWrite

iRelease

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iRelease *icl_resource*

Description

Releases the ownership of a resource that has been reserved by iTake.

Arguments

- *icl_resource*

  A required value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* will be released.

Return value

None.

Example

```
iRelease RegA
iRelease InstA
```

Related Commands

iCall
iMerge
iTake

iReset

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iReset [-sync]

Description

Adds a sequence of actions to the current pattern set that is required to set the ICL network into the reset state.

The exact sequence performed depends on the presence of specific ports in the current design. Each ResetPort is pulsed high and each TRSTPort is pulsed low. If there are TMSPorts without a corresponding TRSTPort, TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low.

After an iReset command, the entire ICL is in its reset state and the TAP finite state machines (when present) are in RunTestIdle state.

You should only use the iReset command once at the beginning of a pattern set. It is illegal to include an iReset inside an instrument PDL as its effect is global and destroys any ongoing action from all instruments. The iReset command is only allowed after open_pattern_set, in an iProc associated with the current design, or in test_setup/test_end procedures (the latter only allowed in contexts other than "patterns -ijtag"). Placing it inside an instrument-level iProc prevents this iProc from being iCalled from a higher level module.

If there is no explicit iReset at the beginning of an open pattern set, an iReset is implied unless the -no_initial_ireset option is used with the open_pattern_set command. Refer to the open_pattern_set command description to understand when and how the -no_initial_ireset option should be used.

Arguments

- -sync

An optional switch that forces a synchronous reset even if there are TRSTPorts in the current design. TCK is pulsed five times with TMS held high followed by an extra pulse of TCK with TMS low. If there is no TMS port, the -sync switch is ignored.

Return Value

None

Example

The following examples set the ICL network into the reset state:

```
iReset
iReset -sync
```

Related Commands

open_pattern_set

iRunLoop

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iRunLoop {count [-tck | -sck [icl_instance_path.]icl_clock_port]} | {-time *tvalue*}

Description

Creates a vector loop of a given duration.

In the generated pattern, the loop is converted into a loop of tester periods.

All clocks defined with the add_clocks commands toggle at their specified period in a free running fashion during the entire pattern. When using the -tck option, the iRunLoop count is expressed in terms of TCK periods and TCK is toggling during the loop. When the -sck [icl_instance_path.]icl_clock_port option is used, TCK is not toggling.

The specified [icl_instance_path.]icl_clock_port must have been specified in a prior iClock command such that its controlling source was verified and its effective cumulative multiplication factor computed. The size of the loop in the pattern depends on the computed effective multiplication factor as well as the ratio between the clock source period and the tester period. Table 7-19 outlines the resulting loop size in the generated pattern based on the various options.

Table 7-19. RunLoop Scaling Algorithm

Condition	Results
open_pattern_set -tck_ratio = 1 iRunLoop X -tck	TCK is defined with a pulsing waveform with the falling edge preceding the rising edge. The loop count is X and is over a single vector where TCK is set to pulse
open_pattern_set -tck_ratio = 2 iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over 2 vectors where TCK is low for the first vector and high for the second one
open_pattern_set -tck_ratio = 4, 8, 16,... iRunLoop X -tck	TCK is defined with a data waveform. The loop count is X and is over <tck_ratio> vectors where TCK is low for the first and last quarter of vectors and high for the middle ones.

Table 7-19. RunLoop Scaling Algorithm

Condition	Results
iRunLoop X -sck <clk> where <clk> has a source that was defined as an asynchronous clock using the add_clocks -period command	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor> * (<async clock period>/<tester period>)
iRunLoop X -sck <clk> where <clk> has a source which was defined as a synchronous clock and the waveform defined for this clock has 2 edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor>
iRunLoop X -sck <clk> where <clk> has a source that was defined as a synchronous clock and the waveform defined for this clock has 2*n edges	TCK is inactive during the loop. The loop count is Y and is over a single vector. Y = X / <effective multiplication factor>*1/n

Arguments

- **count**

  Required integer that specifies the number of clock cycles to be seen on TCK or on [icl_instance_path.]icl_clock_port. It is not required when the -time switch is used.

- -tck

  Optional switch that specifies that count refers to the test clock TCK.

- -sck [icl_instance_path.]icl_clock_port

  Optional switch and value pair that specifies for which clock pin the count is for. The value [icl_instance_path.]icl_clock_port specifies an internal or top-level clock port defined inside the ICL description. The specified clock port must be a port declared as a ClockPort or ToClockPort inside the ICL and it must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description. An iClock command on the [icl_instance_path.]icl_clock_port must have been issued prior to issuing the iRunLoop command.

- -time *tvalue*

  Optional switch and value pair that specifies a time delay *tvalue* instead of a cycle count. When TCK is OFF, iRunLoop -time is converted to a single vector loop, with a loop count of *tvalue*/tester_period rounded up to the next integer. If TCK is free running, TCK is ON and the loop count must be a multiple of the TCK ratio.

Return Value

None

Example 1

The following example specifies a run loop of 10000 tck period. The tck clock is toggling during the loop when the iRunLoop command is using the -tck option.

```
iRunLoop 10000 -tck
```

Example 2

The following example specifies a run loop of a 1000 system clock cycles on the clk pin of the u1 instance. The iClock command checks that the clock port is controlled by a valid source and computes the cumulative multiplication factor between the primary input and the clock port. The run loop is automatically adjusted such that the u1.clk port receives a minimum of 1000 cycles at the u1 instance. The TCK port may or may not be toggling.

```
iClock u1.clk
iRunLoop 1000 -sck u1.clk
```

Example 3

The following examples specify run loops that must last for at least the specified time.

```
iRunLoop -time 2000ns
iRunLoop -time 4.56E+2ns
iRunLoop -time 5.67E-1us
```

Related Commands

add_clocks
delete_clocks
iClock
iClockOverride
report_icl_modules
report_clocks

is_collection

Context: unspecified, all contexts

Mode: all modes

Usage

is_collection **string** [-with_object *obj_type*] [-single_type_objects]

Description

Indicates whether string is a collection handle.

This command returns true if the provided string is a collection handle. You use this command inside a Tcl proc to check if the passed in argument is a collection of objects versus names of objects. See the example below for an example usage.

You can also use this command to check if the collection contains only one type of objects or objects of a specific type.

Arguments

- string

  A required string that specifies the collection whose existence is to be verified. This command returns true if the collection exists.

- -with_object *obj_type*

  An optional switch and string that specifies whether the collection specified by string contains at least one object of the type specified by obj_type. This command returns true if at least one object of the type specified by obj_type exists.

- -single_type_objects

  An optional switch that specifies whether the collection specified by string contains objects of all the same type. This command return true only if all objects are of the same type.

Example

The following example uses the is_collection command to verify that the net argument is already a collection of nets and, if not, the get_nets command is used to create the collection of nets assuming <net> is instead a Tcl list of net names. If the passed-in argument is a collection, it is checked to verify that it only contain net objects. With this code, the proc myProc can be

invoked with either a Tcl list of names of nets, or a collection of nets and the proc automatically adapts.

```
proc myProc {net} {
   if {[is_collection $net]} {
       if {![is_collection $net -with_object net -single_type_objects]} {
           return -code error "<net> contains other object types than net"
       }
   } else {
       set net [get_nets $net]
   }
   #$net is definitely a collection of nets
   ...
}
```

Related Commands

add_to_collection
append_to_collection
compare_collections
filter_collection
foreach_in_collection
index_collection
range_collection
remove_from_collection
sizeof_collection
sort_collection

iTake

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iTake *icl_resource*

Description

Used inside of an iProc to reserve a resource (a port, a register or an instance) for exclusive use with this iProc.

No other parallel running iProcs are allowed to alter states or clock frequencies on the resources taken by an iProc. The reservation persists until the end of the iProc.

Arguments

- *icl_resource*

  A required value that specifies the name of a register, a port or an instance. If an instance is specified, then all registers, ports and instances in *icl_resource* will be taken.

Return value

None.

Example

```
iTake RegA
iTake InstA
```

Related Commands

iCall
iMerge
iPrefix

iUseProcNameSpace

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iUseProcNameSpace [*name_space*]

Description

Specifies a name space that is used for all subsequent iCalls made within the iProc in which the iUseProcNameSpace command appears.

The scope of this command is limited to within the iProc in which it appears. Invoking this command a second time within the same iProc changes the name space to the new value of *name_space*.

If iUseProcNameSpace is invoked without the *name_space* argument, the name space is set to that of the root module.

Arguments

- *name_space*

  An optional string that specifies the name space.

  The name space name must be formed from an initial upper or lower case alphabetic character followed by an optional sequence of upper or lower case alphanumeric or underscore characters ([a-zA-Z][a-zA-Z0-9_]*).

Return Value

None

Example

```
iUseProcNameSpace MyNameSpace //set namespace to MyNameSpace
iUseProcNameSpace // return to root name space
```

Related Commands

get_iproc_argument_default
get_iproc_argument_list
get_iproc_body
get_iproc_list
get_iproc_list
iCall
iProc
iProcsForModule

iWrite

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

Mode: analysis

Prerequisites: In the “patterns -ijtag context”, to use this command you must have previously opened a pattern set with the open_pattern_set command.
In the “dft”, “patterns –scan” and “patterns -scan_retargeting” context, you can use this command only as part of an iProc that is called during the processing of a procfile. For more information, see “IJTAG and ATPG in Tessent Shell” in the *Tessent IJTAG User's Manual*.

Usage

iWrite **[icl_instance_path.]writable_icl_object** value

Description

Adds a write operation to the command queue that will be solved by the next iApply command.

The ICL objects that can be written are DataRegisters, ScanRegisters, DataOutPorts, DataInPorts or an alias of any of these objects. A value to be written must be specified as the second argument. How the iWrite commands are interpreted by the iApply command is described in the iApply command description.

Arguments

- **[icl_instance_path.]writable_icl_object**

  Specifies the ICL objects that are to be written. You can specify to add write operations for the following ICL objects: DataRegisters, ScanRegisters, DataOutPorts, DataInPorts, or an alias of any of these objects. The ICL objects must be found at the computed effective_icl_instance_path. The effective_icl_instance_path is described in detail in the iCall command description.

- value

  An optional constant number represented with Tcl binary, hexadecimal, or integer syntax or with a string corresponding to an enumeration symbol associated to the written objects in the ICL description.

Return Value

None

Example

The following example requests the writable_icl_object mode to be written with the value represented by the enumeration symbol *goNogo* defined in ICL. The two requested write operations are solved when the iApply command is executed.

```
iWrite mode goNogo
iWrite Count 0xa6
iApply
```

Related Commands

iApply
iCall
iPrefix
iRead

launch_sid_tester

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

Mode: setup, analysis

Usage

launch_sid_tester **-cdp** ***cdp_path*** [-new]

Arguments

- **-cdp** ***cpd_path***

  A required switch and string pair that specifies a CDP.

- **-new**

  An optional switch that specifies to create a new CDP at the location specified by the -cdp switch. By default, the tool opens an existing CDP.

Description

Directs Tessent Shell to launch the SID tester process on a local host using the selected_tester and the port number as specified in the configuration data file. Additionally, this command specifies an existing CDP or initializes a new CDP.

During the launch, Tessent Shell passes the hardware and pin map configuration data to the SID tester so that the SID tester can perform the proper setup tasks.

You must have loaded the configuration data file with the read_config_data command prior to running this command.

Note

SVF-based CDPs are configured with asynchronous clocking. The tool does not provide clock waveforms to the DUT, therefore you must source the appropriate clocks yourself.

Note

You can only work with one SID tester at a time. To launch a new SID tester you must first shutdown the current SID tester with the shutdown_sid_tester command.

Example

The following example initializes a new CDP named “cdp1” and launches the SID tester process.

```
launch_sid_tester -cdp ./cdp1 -new
```

Refer to “Executing an IJTAG Test Pattern” in *Tessent SiliconInsight User’s Manual for Tessent Shell* for a sample usage scenario.

Related Topics

read_config_data

shutdown_sid_tester

lock_current_registration

Context: unspecified, all contexts

Mode: all modes

Usage

lock_current_registration

Description

Locks all previously registered commands and attributes preventing them from being unregistered and modified.

After you verified that your registered commands and/or attributes work correctly, you can make them into a *plugin* that Tessent Shell automatically loads during initialization. Simply place your files in a directory called <plugin_dir>/tcl_modules/<my_dir> where <plugin_dir> and <my_dir> are arbitrary directory names. Set the TESSENT_PLUGIN_PATH environment variable to point to the <plugin_dir> directory and any files with a .tcl extension found inside <plugin_dir>/tcl_modules/<my_dir> will be evaluated into the Tcl namespace ::tcl_modules::<my_dir>. Just like the PATH environment variable, you can specify more that one search location by separating multiple directory namies with a colon.

If you name your <plugin_dir> directory name “tessent_plugin” and place it one level above the bin directory of Tessent Shell, it will be automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the command lock_current_registration is issued so that they cannot be unregistered accidently.

Arguments

- **None**

Example

The following example registers a string attribute called myAtt for the object type pin and then issues a lock_current_registration command to make it impossible to unregister the attribute.

```
register_attribute -name myAtt -value_type string -object_type pin
lock_current_registrations
```

Related Commands

register_tcl_command
register_attribute
unregister_attribute
unregister_tcl_command

macrotest

Context: dft -edt, patterns -scan

Mode: analysis

Usage

macrotest {-**MULtiple_macros** ***macro_filename***}
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]
[-VERIfy | -NO_VERIfy] [-MASK_nonobservation_sites]
[-NO_FIll_patterns] [-TE_observation_only | -LE_observation_only]
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]
[-FAILure_analysis] [-EXHAUSTIvely_search]
[-SKIP_PATterns *num_to_skip*] [-FIRST_OBS_pattern]

or

macrotest {{***ID#*** | ***pin_pathname*** | ***instance_name***} ***pattern_filename***}
[-L_H | -NO_L_H] [-VARY_observe_by_pattern] [-VERBose | -NO_VERBose]
[-VERIfy | -NO_VERIfy] [-MASK_nonobservation_sites]
[-NO_FIll_patterns] [-TE_observation_only | -LE_observation_only]
[-NO_REPOrt_observation_candidates | -REPOrt_observation_candidates]
[-FAILure_analysis] [-EXHAUSTIvely_search]
[-SKIP_PATterns *num_to_skip*] [-FIRST_OBS_pattern]

Description

Runs the MacroTest utility to automatically generate manufacturing test patterns for embedded logic and memories referred to as macros. This utility is useful for applying specific patterns that test the insides of a macro, thereby improving overall IC test quality.

Note

Although a macro is typically embedded logic or a memory, it can also be a disjoint set of internal sites or a single block of hardware represented by an instance in HDL.

MacroTest reads patterns from the text file you provide and converts them into scan-based manufacturing test patterns. These test patterns, when applied to the chip, deliver the underlying patterns to the appropriate embedded location through intervening logic. The following is a brief summary of the capabilities of MacroTest:

- Reduces overall test generation time
- Has no impact on area or performance
- Enables you to test multiple macros in parallel
- Supports user-selected scan observation points
- Allows you to define macro output values that do not require observation

- Supports synchronous memories; for example, supports positive (or negative) edge-triggered memories embedded between positive (or negative) edge-triggered scan chains
- Analyzes single macros, and report patterns that are logically inconsistent with the surrounding hardware
- Fault grades the logic surrounding the macro

As a special form of the run command, the macrotest command uses all existing ATPG and cell constraints and limits. By default, if you have added faults, the macrotest command randomly fills bits that are unnecessary for testing the macro and fault simulates. This is recommended so you can observe the pattern conversion progression. You can always issue the reset_state command after the macrotest run to revert to no coverage due to the MacroTest patterns.

Note

To obtain a high degree of precision for resolving any conversion problems, MacroTest fault simulates one pattern at a time, rather than 32 at a time.

The observability of some patterns may not be verified. Pin(s) and pattern are reported and whether or not observability was verified, and then processing stops. Typically, this is because a scan chain clock and an input on the macro are shared. Also, pulsing the macro pin pulses the scan clock, causing the scan cells to update, making the observability invalid. In such cases, specify a sequential MacroTest pattern such that you specify the macro input values to read, including a pulse of the shared clock, in the first cycle of the pattern. Specify the known output values in the second cycle. MacroTest uses the sequential engine for such patterns, allowing a full cycle for the values to come out of the macro and propagate to the scan cells where they will be captured. Such operation of a shared clock should let you verify the macro's observability.

If the macro being tested is a RAM, the MacroTest patterns might perform several writes followed by several reads. In this situation, each write and read becomes a separate scan pattern with its own scan load. If loading the scan chain causes spurious writes into the RAM, Tessent FastScan destroys the data, making the reads access invalid data rather than the data that was written; therefore, the macro should be able to hold state while operating the scan chain to allow a sequence of inputs to be converted to a valid sequence of scan patterns (with a scan load per scan pattern).

For the rare case where macro outputs have paths to both leading edge and trailing edge capturing scan cells, the macrotest command includes switches that let you specify either trailing edge or leading edge observation. When you prepare the MacroTest patterns file, you can specify when to capture or observe so that the pattern matches the selection. For more information, see the -Te_observation_only and -Le_observation_only switches.

For more information on using the macrotest command, see "Using Tessent FastScan MacroTest Capability" in the *Tessent Scan and ATPG User's Manual*.

Note

When testing multiple macros, each macro opens a pattern file. When the number of open files reaches your operating system-specific limit, no more files can be opened. This may force MacroTest to fail. To avoid this problem, you may need to issue the appropriate version of limit, unlimit, or ulimit Linux shell commands before running MacroTest to increase the maximum number of files that can be open at one time. Check your Linux documentation for the appropriate command.

Arguments

- **-MULtiple_macros *macro_filename***

  A switch and string pair that test multiple macros in parallel using the macrotest commands in the *macro_filename* file. The file must contain only macrotest commands, one for each of the macros to test in parallel. Mentor Graphics recommends that you first test each macro individually in a separate run. Individual runs allow analysis if success is not possible for any single macro. When you test each macro successfully, add its corresponding command to *macro_filename* and remove it from the dofile (or command line).

Note

If you successfully tested a macro individually, you can probably test it successfully in parallel. However, Mentor Graphics cannot guarantee this due to the enormous variety of possible embeddings.

If you use the -Multiple_macros switch, you must include in the file all macros you want to be simultaneously tested. You cannot use this switch and also specify a *pattern_filename* or individual macro (by its *ID#*, *pin_pathname* or *instance_name*) on the same command line. You can test one subset of all macros using one -Multiple_macros run, with one *macro_filename* containing those macros and their pattern file specifications, and then test another subset by using another -Multiple_macros run. If you use other arguments with the -Multiple_macros switch, then macrotest uses those arguments as the default arguments for all macros in the file. For an example that uses this switch, see the Examples section at the end of this command description.

- ***ID#***

  An integer that specifies the gate identification number of the object to use in the macrotest run. The value of the *ID#* argument is the unique identification number that is automatically assigned to a gate within the design during the model flattening process.

- ***pin_pathname***

  A string that specifies the name of the input or output pin of an ATPG library model to use in the macrotest run.

- ***instance_name***

  A string that specifies the name of the instance to use in the macrotest run.

- ***pattern_filename***

  A string that specifies the name of the file containing the set of patterns to be applied to the macro.

- **-**L_H

  An optional switch that represents {L,H} as {LO,HI} output values in the macro patterns file. An error is issued if a 0 or 1 output value is specified in the macro patterns file. This is the default.

- **-**NO_L_H

  An optional switch that uses {0,1} to specify {LO,HI} output values in the patterns file, rather than the default {L,H}. If the default {L,H} is used, checking is done to ensure that the pin direction matches (output pin for L,H; input pin for 0,1). If the option -NO_L_h is issued, no such checking is possible.

- ***-VARY_observe_by_pattern***

  An optional switch that, if issued, causes MacroTest to assume that a macro may need to vary its observation path or change where it is observed at after the first successful observation path is discovered by MacroTest on the first pattern with known outputs. The default is to assume the same path can be reused without calling ATPG to verify that on each pattern.

- **-**VERBose

  An optional switch that causes all informative messages to be issued. Use this option when testing any macro for the first time, so that all information about the progress and possible issues are conveyed. This is the default.

- **-**NO_VERBose

  An optional switch that turns off the default verbose output. When using this option, important warning and informative messages may be left out.

- **-**VERIfy

  An optional switch that causes one extra simulation per pattern to verify each macro's controllability (the macro input values) and observability (the values captured into a scan cell or observed at a chip pin). The observability check verifies that complementing all macro outputs causes each scan cell or chip pin where observation is occurring to change its value if a defect caused the observed macro output to be faulty. This is the default, and you should never turn off this check to solve problems of failure to verify a pattern, which will halt the macrotest run. Instead, you must debug the issue causing the failure. Failure to verify control values typically causes simulation or tester mismatches. Failure to observe typically means that defects in the macro cannot be detected on a tester, which defeats the entire purpose of creating macrotest patterns.

- **-**NO_VERIfy

  An optional switch that refrains from performing an extra simulation per pattern to verify that the macro input values were delivered by the pattern, and that changing the macro

outputs changes the observation sites. This switch should never be used rather than investigating the cause of verification failure. Failure to control of observe the macro will either make the pattern useless, or will predict the wrong value and halt the tester.

- -MASK_nonobservation_sites

  An optional switch that places an X in all scan cells except the ones used for observation on a pattern-by-pattern basis.

  You need to specify both the -NO_FIll_patterns and the -MASK_nonobservation_sites switches to force only the scan cells used by macrotest to observe macro outputs to have known values for the scan unload. If you specify the -NO_FIll_patterns switch without also specifying the -MASK_nonobservation_sites switch, all values that propagate to scan cells are captured, regardless of whether they are from the macro outputs or from the known PIs and scan cells that fanout to both the macro inputs and other scan cells; that is, scan cells other than those observing macro outputs may have known scan-out values.

- -NO_FIll_patterns

  An optional switch that causes the unspecified bits in scan chains to remain at a value X, so that only the values needed to test the macro appear as known values. Using this option, you can see which scan chain and primary input values are needed to test the macro by using the write_patterns command. You can use the -MASK_nonobservation_sites option to convert captured values that are not needed but known to X in the scan-out.

- -TE_observation_only

  An optional switch that observes at only one of the trailing edge (TE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- -LE_observation_only

  An optional switch that observes at only one of the leading edge (LE) sites when a macro has both trailing edge and leading edge observation sites. The default is to observe at any site.

- -NO_REport_observation_candidates

  An optional switch that refrains from reporting the reachable observation sites. In some cases, the number of possible scan cells that an output can go to is large and the report long, so the default is not to report the candidates.

- -REport_observation_candidates

  An optional switch that reports the reachable observation sites, and whether those sites are already eliminated as observation candidates due to inability to capture. For example, a cell constraint to observe X prevents capture. If you use this switch, a list of possible observation sites (candidates) per macro output pin is displayed at the beginning of the macrotest run. The -Report_observation_candidates switch is sometimes useful for discovering that different macros in a multiple macro run have the same observation site possibilities. It also helps explain why it may be impossible to observe both macros at the same time.

The observe_at syntax in the macrotest patterns file allows you to specify where to observe one or more of the outputs of a macro for all patterns. The -Report_observation_candidates switch is useful for selecting the name(s) to use as observation site(s).

Note

Usually, it is better to let MacroTest decide where to observe, but to quickly test many macros at once, correct guidance can reduce the runtime. Incorrect guidance leads to failure to succeed where success might have been possible. Mentor Graphics recommends that you let MacroTest decide where to observe rather than restrict its choices. However, restricting observation may allow MacroTest to provide a more focused analysis of failing situations.

- -FAILure_analysis

  An optional switch that specifies the tool to determine why ATPG cannot succeed. Because this failure analysis can consume significant processing time, the default is to avoid failure analysis for rapid debug iteration. You should use this switch only as a last resort to determine why the embedding prevents justification of the macrotest pattern that causes ATPG to fail.

- [-EXHAUSTIvely_search]

  An optional switch that performs an exhaustive search. For some situations, an exhaustive search is needed; however, MacroTest can run for cpu weeks when hundreds of macros are tested together in one run. This switch only solves problems where one macro interacts with another macro; this switch is not used for problems where a single macro conflicts with its own needed boundary values and/or netlist requirements surrounding that macro. By default, this switch is omitted due to the infrequency of such debilitating macro interactions and the typical runtime cost to find them if they occur.

- -SKIP_PATterns *num_to_skip*

  An optional switch and integer pair that skips the first *num_to_skip* patterns. This option is used for analyzing problems after pattern creation when a MacroTest run fails. For example, if the third pattern fails to be converted (the transcript shows two patterns converted and simulated and then reports a failure), specify 2 for *num_to_skip*, so that MacroTest skips the first two patterns and starts on the third pattern. This enables you to quickly iterate on the problematic pattern. The -SKIP_PATterns *num_to_skip* option is not used for production patterns.

- -FIRST_OBS_pattern

  An optional switch used only for analysis when MacroTest fails. This option skips all of the control-only patterns and processes the first pattern with known (observable) macro outputs as if it were the first pattern in order to speed iterations and get to the problematic patterns sooner.

Example

The following example tests all of the macros in parallel (those with a macrotest command in the “macrofile” file). The example changes the default, whereby output 0 values in MacroTest patterns are indicated by L, and output 1 values are indicated by H. Instead, 0 and 1 are used respectively. Note that macrotest commands inside the macrofile can override the -NO_L_H switch below by selecting an alternative option. For example, one or more of the commands could use -L_H if those macros have files that use L and H to indicate 0 and 1 output values.

```
macrotest -multiple_macros macrofile -NO_L_H
```

For more examples, refer to “Using Tessent FastScan MacroTest Capability” in the *Tessent Scan and ATPG User’s Manual*.

Related Commands

add_faults
set_decision_order
set_contention_check
set_gate_report
set_macrotest_options

mark_display_instances

Context: all contexts

Mode: setup, analysis

Usage

mark_display_instances {***gate_id#*** | ***instance_name*** | ***pin_pathname***}... | **-All** | **-Selected**}
[-marking_index {1 | 2 | 3 | 4 |5}] [-Display {DEBug | DESign}]

Description

Changes the color of the specified instances in the schematic windows.

You can specify up to five different colors to use in marking. The color difference can help you to quickly locate and identify particular instances in a complex schematic view. The color difference remains until you use the unmark_display_instances command to return the gate to its unmarked color.

Tip: If a gate is selected, the color applied by the mark_display_instances command is not visible until the gate is unselected.

You can change the colors used for marking by choosing **Edit > Preferences** in the DFTVisualizer and clicking the **Colors** tab. For more information, refer to "Setting DFTVisualizer Preferences" in the *Tessent Shell User's Manual*.

When you mark an object(s) by entering the mark_display_instances command into the command line (as opposed to using menus or Ctrl-M), the color assigned to Color Index 1 in the DFTVisualizer Preferences dialog box is used to mark the object(s). By default, Green is assigned to Color Index 1. For more information, refer to "Colors Preferences Dialog Box" in the *Tessent Shell User's Manual*.

Arguments

- ***gate_id#***

  A repeatable integer that specifies the gate identification number of an instance to mark. The value of the *gate_id#* argument is the unique identification number assigned to every gate in the design during the model flattening process.

- ***instance_name***

  A repeatable string that specifies the name of an instance to mark.

- ***pin_pathname***

  A repeatable string that specifies the name of a pin to mark.

- **-All**

  A switch that specifies to mark all the gates displayed.

- **-Selected**

  A switch that specifies to mark all currently selected gates.

- -marking_index {1 | 2 | 3 | 4 |5}

  An optional switch and literal pair that specifies the color to be used to mark objects in DFTVisualizer windows. By default, index 1 is used to mark objects. You can specify the color associated with each index from the Colors tab of the DFTVisualizer Preferences dialog box when Marked is selected. For more information on marking, see "Customizing Marking Colors in the Schematic Windows" and "Marking and Unmarking Objects in the Windows".

- -Display {DEBug | DESign}

  A switch and literal pair that specifies the window for which to apply the mark_display_instances command. The following literals can be used with the -Display switch:

  DEBug
  Applies the settings to the Debug window. This is the default behavior.

  DESign
  Applies the settings to the Design window.

Examples

The following example marks two instances using the color set as Color Index 3, specifying the instance name of one and the gate identification number (85) of the other:

```
mark_display_instances /i$142/q 85 -marking_index 3
```

Related Commands

select_display_instances

unmark_display_instances

migrate_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

migrate_layout ***ldb_directory*** **-ladb** ***preexisting_ldb*** [-vdb *preexisting_vdb*]

Description

Generates an LDB directory hierarchy that contains the specified preexisting LDB.

Use this command to copy a preexisting LDB into the LDB directory hierarchy that became available with the Tessent Diagnosis 2013.2 release. This command allows you to upgrade to the LDB directory without regenerating the LDB from the LEF/DEF files. The tool automatically copies the preexisting LDB into the new LDB directory.

For the DFM diagnosis flow, use this command to additionally copy a preexisting violations database (VDB), which was generated automatically during DFM diagnosis, into the new LDB directory.

Arguments

- ***ldb_directory***

  A required string that specifies the pathname to a new LDB directory.

- **-ladb** ***preexisting_ldb***

  A required switch and string pair that specifies the pathname to a preexisting LDB.

- **-vdb** ***preexisting_vdb***

  An optional switch and string pair that specifies the pathname to a preexisting VDB.

Example

The following example generates a LDB directory named mylayoutdb. It then copies the example2.ldb and example_dfm.vdb databases into the LDB directory.

```
migrate_layout newlayoutdb -ladb preexisting.ldb -vdb preexisting_dfm.vdb
```

move_config_element

Context: dft

Mode: analysis

Usage

move_config_element ***object_spec***
[-in_wrapper *wrapper_name_or_object*]
[-before *before*| -after *after* | -first | -last]
[-replace]

Description

Moves a configuration element from one location to another.

Repeatable and non-repeatable wrappers and repeatable properties can be moved from one wrapper to another. Single wrappers and properties can only be moved within their parent wrapper.

Arguments

- ***object_spec***

  A value that specifies the name of a configuration element or a collection containing one configuration element to be moved. When using the -in_wrapper option, the name is the relative pathname with respect to the specified wrapper. When not using the -in_wrapper option, the name is the complete pathname with respect to the root of the partition. To move an element from one wrapper to the next, you must use the get_config_elements or the "get_config_value -object" command to get the object and use it as the object_spec. The moved element must not already exist in the destination wrapper unless the -replace option is used, in which case the existing element will be deleted before being replaced by the moved one.

  When the allowed content of the destination parent wrapper is defined with Wrapper, the metadata definition of the source and destination element must be identical (both definition must reference the same metadata definition). If this is not the case, an error message is generated.

- -in_wrapper *wrapper_name_or_object*

  An option value pair that specifies that the configuration element is to be moved inside a specific wrapper. The *wrapper_object_spec* can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by get_config_elements or "get_config_value -object". When the -in_wrapper is not specified, the destination parent wrapper is assumed to be the same as the source parent wrapper unless the -before or -after option is specified.

- -before *before*

  An optional option value pair that specifies the configuration element in front of which to move the new configuration element. When the -before option is used with a collection

containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

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

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

- -after *after*

  An optional option value pair that specifies a configuration element after which to move the new configuration element. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used because the parent wrapper is extracted from the object.

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

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

- -first

  An optional option that specifies to move the new configuration element before any other element in the parent wrapper.

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

- -last

  An optional option that specifies to move the new configuration element after all other elements in the parent wrapper.

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

- -replace

  An optional option that suppresses the error that is normally generated when moving a configuration element from one wrapper to another and the leaf_name conflicts with an existing element in the destination wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Example

The following example moves configuration elements within and across parent wrappers.

```
> set_transcript_style input_only
> set dft [add_config_element DftSpecification(moda,rtl)]
{/DftSpecification(moda,rtl)}
> set ijtag [add_config_element IjtagNetwork -in_wrapper $dft]
{/DftSpecification(moda,rtl)/IjtagNetwork}
> set host [add_config_element HostScanInterface(1) -in_wrapper $ijtag]
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)}
> set sib1 [add_config_element sib(1) -in_wrapper $host]
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(1)}
> set sib2 [add_config_element sib(2) -in_wrapper $host]
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(2)}
> set sib3 [add_config_element sib(3) -in_wrapper $host]
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(3)}
> set sib4 [add_config_element sib(4) -in_wrapper $host]
{/DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(4)}
> report_config_data $dft

DftSpecification(moda,rtl) {
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(1) {
      }
      Sib(2) {
      }
      Sib(3) {
      }
      Sib(4) {
      }
    }
  }
}
> move_config_element $sib3 -in_wrapper $sib2
> report_config_data $dft

DftSpecification(moda,rtl) {
  IjtagNetwork {
    HostScanInterface(1) {
      Sib(1) {
      }
      Sib(2) {
        Sib(3) {
        }
      }
      Sib(4) {
      }
    }
  }
}
> #Because -in_wrapper is not specified, $sib one is moved within its
> #parent wrapper
> move_config_element $sib1 -last
> report_config_data $dft

DftSpecification(moda,rtl) {
```

```
      IjtagNetwork {
        HostScanInterface(1) {
          Sib(2) {
            Sib(3) {
            }
          }
          Sib(4) {
          }
          Sib(1) {
          }
        }
      }
    }
    > #example showing the use of a config element name in the
    > #-in_wrapper option
    > move_config_element $sib1 -in_wrapper \
    /DftSpecification(moda,rtl)/IjtagNetwork/HostScanInterface(1)/Sib(2)
    > report_config_data $dft

    DftSpecification(moda,rtl) {
      IjtagNetwork {
        HostScanInterface(1) {
          Sib(2) {
            Sib(3) {
            }
            Sib(1) {
            }
          }
          Sib(4) {
          }
        }
      }
    }
    > #Because -in_wrapper is not specified, $sib one is moved within its
    > #parent wrapper
    > move_config_element $sib1 -first
    > report_config_data $dft

    DftSpecification(moda,rtl) {
      IjtagNetwork {
        HostScanInterface(1) {
          Sib(2) {
            Sib(1) {
            }
            Sib(3) {
            }
          }
          Sib(4) {
          }
        }
      }
    }
```

Related Commands

add_config_element

move_connections

Context: all contexts

Mode: insertion

Usage

move_connections **-from *obj_spec1* -to *obj_spec2*** [-net_name *net_name*] [-silent]

Description

Moves connections from pin or net objects to pin objects.

The -from obj_spec1 argument is assumed to be a net when the -to obj_spec2 argument is one level below the hierarchy of obj_spec1. For example, the following command moves the u1/a pin to the *net* by the name of u1/a, instead of the pin, because u1/u2/b is below u1.

```
move_con -from u1/a -to u1/u2/b
```

In another example, u2/b is not below u1. In this case, the command creates the connection to the u1/a *pin* instead of to the net.

```
move_con -from u1/a -to u2/b
```

The specific behavior of the move_connections command varies based on the object type found inside obj_spec1.

If the object within obj_spec1 is:

- a pin

  The net connected to this pin is moved to the pin specified in obj_spec2 and the pin in obj_spec1 is left unconnected.

- a net that is also an input port

  The net is renamed everywhere it is used and is then connected to the pin in obj_spec2. The input port is left with no fanout. The renamed net name is the specified -net_name value or the original net name, when the -net_name option is not specified, uniquified using the _ts# suffix.

- a net that is also an output port

  The net is renamed everywhere it is used and is then connected to the pin in obj_spec2. The output port is left with no fanin. The renamed net name is the specified -net_name value or the original net name, when the -net_name option is not specified, uniquified using the _ts# suffix.

- a net that is not also a port

  The net is renamed everywhere it is used, except at its driver source, and is then connected to the pin in obj_spec2. The driver source is left connected to the original net with no fanout. The renamed net name is the specified -net_name value or the original net name, when the -net_name option is not specified, uniquified using the _ts# suffix.

The obj_spec1 must be a single pin or net name, a Tcl list of pin or net names, or a collection of pin or net objects. The obj_spec2 must be a single pin name, a Tcl list of pin names or a collection of pin objects. The obj_spec1 and obj_spec2 must have exactly the same size. The -from and -to objects must exist in the same instance; a move is not allowed across hierarchy.

This command returns a collection containing the net object that ends up being connected to the pin object in obj_spec2. If the -from object is a pin, the command returns the nets that were moved. If the -from object is a net, the command returns the nets that were created.

If an object does not exist in obj_spec1 or obj_spec2 and the -silent option is specified, the object is ignored and the move action skipped. If an object does not exist and the -silent option is not specified, the tool generates an error.

You must be careful when using the move_connections command inside a non-uniquified design. You must only do the move or disconnection once per module and once per generate loop count. See how the parent_instance and the leaf_name_hash attributes are used in section 3 of Example 5 in the get_dft_cell command description to perform this robustly.

Arguments

- **-from *obj_spec1***

  A required switch and value pair that specify the object to be moved. The obj_spec1 must be a single pin or net name, a Tcl list of pin or net names, or a collection of pin or net objects.

- **-to *pinObj2***

  A required switch and value pair that specifies the pin to which the net is moved to. The obj_spec2 must be a single pin name, a Tcl list of pin names, or a collection of pin objects. The obj_spec1 and obj_spec2 must have exactly the same size. The -from and -to objects must exist in the same instance; a move is not allowed across hierarchy.

- -net_name *name*

  An optional argument that specifies the base net name to create when moving a net. The new net name is the specified -net_name value or the original net name when the -net_name option is not specified, uniquified using the _ts# suffix.

- -silent

  An optional argument that specifies to ignore specified objects that do not exist and to ignore the move connection without generating an error message. The returned net object is null.

Example 1

This following example moves the net connected to input pin u1/a to the input pin u2/b. After the operation, pin u1/a is left with no driver.

```
move_connections -from u1/a -to u2/b
```

Example 2

The following example shows how to intercept an input pin from the inside of the instance. By using the get_net introspection command, the net named u1/u2/a is returned instead of the pin as in the previous example. The parent_instance of the net automatically returns u1/u2 which results in the intercepting gate being placed inside the module as required.

```
set net [get_net u1/u2/a]
set gate_inst_name [lindex [get_attribute_value_list $net -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $net -to [get_pins y -of_instance $gate_inst]
create_connections $net [get_pins a -of_instance $gate_inst]
```

Figure 7-17. Intercepting an Input Pin From Inside An Instance

Example 3

This following example shows how to intercept an output port from the inside of the current design. The parent_instance of the port automatically returns a null value which results in the intercepting gate being placed inside the root module as required. The net names created by the move_connections and create_connections commands are derived from the net named being moved or connected. You can use the -net_name option if you want a specific net name to be used instead.

```
set port y
set gate_inst_name [lindex [get_attribute_value_list $port -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod mygate -allow_inst_uniquification]
move_connections -from $port -to [get_pins a -of_instance $gate_inst]
create_connections $port [get_pins y -of_instance $gate_inst]
```

Figure 7-18. Intercepting an Output Port From Inside the Current Design

Example 4

This following example shows how to intercept an input pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new uniquified name. The net that was connected to the pin u1/u2/a ends up connected to pin “a” of $gate_inst. Following the move_connections operation, pin u1/u2/a is left open. The create_connections command is then used to reconnect it to the output of the intercepting gate.

```
set pin u1/u2/a
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins a -of_instance $gate_inst]
create_connections $pin [get_pins y -of_instance $gate_inst]
```

Figure 7-19. Intercepting an Input Pin From Outside An Instance

Example 5

The following example shows how to intercept an output pin from the outside of the instance. This example also shows the use of the parent_instance attribute to properly determine the location of the intercepting gate that must be placed in the same hierarchy level as the intercepted net. Notice that the return value of the create_instance command is used to update the gate_inst value in case the original name already existed inside the design, and the create_instance command created a new uniquified name. The net that was connected to pin u1/u2/y ends up connected to pin y of $gate_inst. Following the move_connections, pin u1/u2/y is left open. The create_connections command is then used to reconnect it to the input of the intercepting gate.

```
set pin u1/u2/y
set gate_inst_name [lindex [get_attribute_value_list $pin -name parent_instance] 0]/mygate
set gate_inst [create_instance $gate_inst_name -of_mod and2 -allow_inst_uniquification]
move_connections -from $pin -to [get_pins y -of_instance $gate_inst]
create_connections $pin [get_pins a -of_instance $gate_inst]
```

Figure 7-20. Intercepting an Output Pin From Outside An Instance

Related Commands

create_connections

delete_connections

no_transcript

Context: unspecified, all contexts

Mode: all modes

Usage

no_transcript { ***command1 command2 ...*** }

Description

Turns off transcripting for the specified commands.

This command is useful when you are writing Tcl procedures in which you do not want your embedded commands to show up in the transcript or log file. The no_transcript command is also not transcripted.

Arguments

- ***command1 command2 ...***

  A required string that specifies the commands that you do not want to appear in the transcript or log file.

Example

The following excerpt from a Tcl procedure (see get_dft_cell, example 5, for the full Tcl procedure), shows a usage example for the no_transcript command:

```
#####################################################################
### Section 5: Retrieves leaf instance name of gate if node is a
### repeated module instance and skips the interception
#####################################################################

if {[info exists insertion_parent_leaf_instance_array($key)] && \
      [get_insertion_option -auto_uniquify_edited_modules] eq "off"} {
   set leaf_cell_inst "$insertion_parent_leaf_instance_array($key)"
   if {$leaf_block_id ne ""} {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_block_id}.${leaf_cell_inst}"]
   } else {
      set cell_inst [get_instance \
         "${parent_instance_name}/${leaf_cell_inst}"]
   }
} else {
   if {$leaf_block_id ne ""} {
      set cell_inst_name \
      "${parent_instance_name}/${leaf_block_id}.insertion_${cell_sel}"
   } else {
      set cell_inst_name \
         "${parent_instance_name}/insertion_${cell_sel}"
   }
   no_transcript {
      set cell_inst [create_instance $cell_inst_name -of_module \
         $dft_cell -allow_instance_uniquification]
   }

   set leaf_name [lindex [get_attribute_value_list $cell_inst -name \
      leaf_name] 0]
   set leaf_block_id [lindex [get_attribute_value_list $cell_inst \
      -name leaf_block_id] 0]
   if {$leaf_block_id ne ""} {
      # Remove the block_id part from the leaf instance name
      set leaf_name [string range $leaf_name \
         [string length $leaf_block_id] end]
   }
   set insertion_parent_leaf_instance_array($key) $leaf_name

   if {$insertion_type == "sink"} {
      no_transcript {
         move_connections -from $node_object \
            -to [get_pins ${input0} -of_instances  $cell_inst]
         create_connections [get_pins ${output} -of_instances \
            $cell_inst] $node_object
      }
   } else {
      no_transcript {
         move_connections -from $node_object -to [get_pins ${output} \
            -of_instances $cell_inst]
         create_connections $node_object [get_pins ${input0} \
            -of_instances  $cell_inst]
      }
```

```
         }
      }

      if {$connect_in2} {
         no_transcript {
            create_connections $in2_object [get_pins $input1 -of_instances \
               $cell_inst]
         }
      }

      if {$connect_sel} {
         no_transcript {
            create_connections $sel_object [get_pins $select -of_instances \
               $cell_inst]
         }
      }
      return $cell_inst
   }
```

open_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

Layout-Aware Diagnosis Flow

open_layout ***layout_database_name*** [-VERIfy [-EXTRA_LAYout_hierarchy ***extra_levels_of_hierarchy_in_layout***> | -EXTRA_DESign_hierarchy ***extra_levels_of_hierarchy_in_design***>]] [-THreshold ***percentage_match***]

SVDB Layout Marker Flow

open_layout **-SVDB** ***svdb_directory_name*** [-32]

Description

Layout-Aware Diagnosis Flow

Opens an existing Tessent Diagnosis tool-compatible Layout Database directory.

Any subsequent usage of this command first closes any existing layout before Tessent Diagnosis opens a new Layout Database. See "Layout-Aware Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information on this flow.

Layout-Aware Diagnosis Flow with DFM

Opens an existing Tessent Diagnosis tool-compatible Layout Database and if this database contains imported DFM information, the layout-aware diagnosis results will be annotated with DFM hit results.

For more information, see "Diagnosis for Design for Manufacturability Analysis."

SVDB Layout Marker Flow

Checks out a *calibreqdb* license and invokes the 64-bit Calibre Query Server software in the background on workstations supporting 64-bit. Otherwise, Tessent Diagnosis executes this command in 32-bit mode. Before using this command, you must have a valid SVDB directory containing net name information and set the CALIBRE_HOME environment variable to your Calibre software tree.

Arguments

- ***layout_database_name***

  A required string that specifies the name of the existing Layout Database directory created with the create_layout command.

Layout-Aware Diagnosis Flow

- -VERIfy

  An optional switch that specifies to perform verification if the current design has not been validated against the current Layout Database. Using this option ensures that in subsequent runs the layout database can be opened for the current design without having to re-run verification.

  At the time of opening the layout, Tessent Diagnosis checks whether the current design has been verified against the current Layout Database. If the current design has not been validated, it performs verification. During verification, layout rule violation information is added to the layout database.

  For each flat model used during diagnosis, you must first perform the verification step. You can do this with the open_layout -verify command. With subsequent diagnosis runs using a verified flat model/LDB combination, you can skip the verification step by just specifying open_layout. If you need to verify many flat models against the same LDB, you can use open_layout -verify to verify them in parallel in separate diagnosis sessions.

  Layout verification rules can be divided into two broad categories: those that check the consistency of LEF/DEF files themselves, and those that match design information with the layout. Only the second type of layout verification rules are re-evaluated when a layout database is verified against a new design. Because the LEF/DEF layout information has not changed, the first type of rules do not need to be re-run.

- -EXTRA_LAYout_hierarchy ***extra_levels_of_hierarchy_in_layout***

  An optional switch and string pair that removes extra levels of layout hierarchy such that the design hierarchy and layout hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if your design is a subset of a larger layout hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

  This switch can only be used in conjunction with the -verify switch.

- -EXTRA_DESign_hierarchy ***extra_levels_of_hierarchy_in_design***

  An optional switch and string pair removes extra levels of design hierarchy such that the layout hierarchy and design hierarchy match. You can only specify this switch once, otherwise the tool issues an error and exits.

  You use this switch if your layout is a subset of a larger design hierarchy. You can specify multiple levels of hierarchy in one string with no spaces.

  This switch can only be used in conjunction with the -verify switch.

- -THreshold ***percentage_match***

  An optional switch and integer pair specifies the minimum percentage match threshold required to successfully complete a layout verification of the LEF/DEF files, and subsequent verification of the layout database. The default threshold value is 85 percent.

SVDB Layout Marker Flow

- **-SVDB *svdb_directory_name***

  A required switch and string pair that specifies the name of a valid SVDB directory. The SVDB must contain net name information.

- -32

  An optional switch invoking the 32-bit version of the Calibre Query Server software. The 64-bit version is the default.

Example 1

In the following example, Tessent Diagnosis loads the existing Layout Database (*design.layout*) into memory:

```
open_layout design.layout
```

Example 2

The following example illustrates using the -VERIfy switch with the open_layout command for two designs: *DesignA.flat* and *DesignB.flat*. Except for different pin constraints, the two designs are essentially identical. Because a high percentage of the logic overlaps, both designs use the same Layout Database for layout-aware diagnosis.

To perform layout-aware diagnosis on these two designs using the same Layout Database, you do the following:

1. Create and verify the Layout Database, in this example *DesignA_layout_database*, with *DesignA.flat* using the create_layout command.
2. Invoke Tessent Diagnosis in scan diagnosis mode and read in the *DesignB.flat* flat model with the read_flat_model command.
3. Using the open_layout command and -VERIFy switch, open the *DesignA_layout_database* with the DesignB.flat model as in the following:

   ```
   open_layout -verify DesignA_layout_database
   ```

   This verifies *DesignB.flat* against the newly-created *DesignA_layout_database*.
4. Proceed with layout-aware diagnosis with *DesignB.flat*.

Related Commands

close_layout
create_layout
delete_layout_verification
report_layout_rules

open_pattern_set

Context: patterns -ijtag

Mode: analysis

Usage

open_pattern_set **pattern_set_name [**-timeplate *timeplate* **|** -tester_period *tester_period***]** **[**-tck_ratio *tck_ratio***] [**-no_initial_ireset**] [**-replace**]**

Description

Opens an empty named pattern set and makes it ready to be populated with the specified PDL commands.

A populated pattern set is closed using the close_pattern_set command. An existing pattern set cannot be overwritten unless it is first deleted or the -replace option is used.

PDL commands can only be specified when a pattern set is open with the exception of the iReset and iCall commands which can also be specified within the test_setup and test_end procedures in the patterns -scan context.

You use the -timeplate switch to specify the timeplate for this pattern set. Specifying a different timeplate is rarely needed as the default timeplate is adequate for standard ijtag operations. Tessent Shell automatically defines the appropriate timeplate based on the specified off value for TCK and the specified -tck_ratio option value. The only time you need a different timeplate is in the rare situation you want to define clock waveforms with more than two edges.

The -tck_ratio option defaults to 1. You must specify a larger value if you have synchronous clocks (add_clocks commands without the -period option). In this case, you specify -tester_period to match your system clock requirements and specify a -tck_ratio value such that TCK toggles at a rate that is not higher than it was designed for.

If all your system clocks are asynchronous (add_clocks commands with the -period option), allow -tck_ratio to default to 1 and specify -tester_period to match your TCK period requirements. Figure 7-21 through Figure 7-28 illustrate the TCK timing and the position of the ScanOut strobing relative to the rising edge of TCK. Notice how, independent of the off value for TCK and the specified tck_ratio value, ScanOut is always strobed one percent of the TCK period before the rising edge of TCK.

With a custom timeplate, you can specify different timing for the measurements and forces of ScanIn, ScanOut and TCK; however, the measurement time for ScanOut must be before the rising of TCK and after the falling edge of TCK when the off value for TCK is 1. For more information about timeplates, refer to "Timeplate Definition" in the *Tessent Shell User's Manual*.

Figure 7-21. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 0

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.24 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Figure 7-22. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 0

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.48 * tester_period);
   f_time = (0.5 * tester_period)
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK f_time;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Figure 7-23. Scan Frame Timing with tck_ratio = 4 and TCK Off Value of 0

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK 0;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Figure 7-24. Scan Frame Timing with tck_ratio >= 8 and TCK Off Value of 0

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0 ;
   measure_po strobe_1;
   force tck 0;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Figure 7-25. Scan Frame Timing with tck_ratio = 1 and TCK Off Value of 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.74 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   pulse_clock t_time t_width;
```

Figure 7-26. Scan Frame Timing with tck_ratio = 2 and TCK Off Value of 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.48 * tester_period);
   f_time = (0.5 * tester_period)
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0 ;
   measure_po strobe_1;
   force TCK f_time;
   pulse_clock t_time t_width;
```

Figure 7-27. Scan Frame timing with tck_ratio = 4 and TCK Off Value of 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force TCK 0;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Figure 7-28. Scan Frame timing with tck_ratio >= 8 and TCK Off Value of 1

```
timing_variables =
   tester_period = 10;
   strobe_1 = (0.96 * tester_period);
   t_time = (0.25 * tester_period);
   t_width = (0.5 * tester_period);
end;

timeplate tessent_ijtag =
   force_pi 0;
   measure_po strobe_1;
   force tck 0;
   pulse_clock t_time t_width;
   period tester_period;
end;
```

Arguments

- **pattern_set_name**

  A required string that defines the name of the pattern set. This name uniquely identifies the pattern set for later use with write_patterns. This name also specifies the name of the iProc, if the pattern set is written out in -pdl format.

Note

The pattern set names “test_setup” and “test_end” are not allowed.

- -timeplate *timeplate*

  An optional switch and string pair that specifies the name of a valid timeplate, defined in a procedure file reference with the set_procfile command. This option is mutually exclusive with the -tester_period option as the tester_period is specified within the timeplate when the -timeplate option is used.

- -tester_period *tester_period*

  An optional switch and value pair that specifies the tester period when the built-in timeplate, shown in Figure 7-21 through Figure 7-28, is used. The default value is 100.0 ns. The default unit when omitted is nanosecond.

  Note that you can use this switch to specify that the Verilog testbench has a precision smaller than its resolution. For more information, refer to the description of the “set time scale” statement in the *Tessent Shell User’s Manual*.

- -no_initial_ireset

  An optional switch that specifies to create a pattern set that is to run after another one while keeping power and clocks active between them. You may need to create a pattern set of this type if you need to insert some ATE actions in the middle of a test such as changing a voltage. These types of pattern sets are exported together when writing out the Verilog test bench and as separate STIL or WGL files when exporting the pattern files to use on the ATE. You should use special care when running these patterns on the ATE because the lack of a reset at the beginning of the pattern makes it impossible to quit in the middle of the pattern and resume from the start when the capture memory is filled up. Data logging on such patterns can only be done in one pass and to the extent of the size of the capture memory on the ATE equipment.

  The -no_initial_ireset option cannot be used on the first open_pattern_set command of a session. When used on a subsequent open_pattern_set command, the end ICL state of the previous pattern_set is used as the beginning ICL state for the current pattern set.

- -tck_ratio *tck_ratio*

  An optional switch and value pair that specifies the ratio between the required TCK period and the tester_period used for this pattern set. The tester period is the value specified with the -tester_period option when using the built-in timeplate or the period entry of the timeplate referenced by the -timeplate option. The default value is 1.

- -replace

  An optional switch that specifies to replace an existing pattern set with the same name.

Return value

None

Example

The following example opens a pattern set called pat1 using the built-in timeplate described in Figure 7-21. The tester_period is over-written to 50 ns. The same pattern set is later replaced by one where the timeplate TP1 is used. TP1 is a timeplate defined inside a procfile and referenced by the set_procfile command in setup mode.

```
open_pattern_set pat1 -tester_period 50ns
iCall myProc
close_pattern_set

open_pattern_set pat1 -timeplate TP1 -replace
```

Related Commands

close_pattern_set
get_open_pattern_set
report_pattern_sets
reset_open_pattern_set

open_tsdb

Context: all contexts

Mode: all modes

Usage

open_tsdb *directory_path_list* [-silent]

Description

Makes the content of TSDB directory visible to the tool. The specified TSDB output directory is automatically opened and does not need to be opened with this command.

Many actions are taken on the opened TSDBs at various steps in the design flow. Some of those actions are described here:

- When the set_current_design command is invoked, the tool automatically reads in the most recent <module_name>.<format>_interface file using the read_verilog or read_vhdl -blackbox option if the corresponding <module_name> does not already exist in memory. This enables you to only load the modules associated to the current design and have the interface view of all child physical or sub blocks loaded automatically.
- All dft_inserted_designs and instruments sub-directories are automatically searched and ICL files corresponding to design modules are automatically loaded after design elaboration.
- The is_physical_module attribute is automatically set on all modules found in a TSDB that were declared as physical blocks using the set_design_level command when they were processed by the tool.

Arguments

- *directory_path_list*

  A required value containing an absolute or relative path to a TSDB directory, or a Tcl list containing one or many absolute or relative paths to TSDB directories. The tool verifies that each specified directory path is an existing TSDB directory and that only one copy of instrument and dft_inserted_design directories exists for any given module/design_id combination in all opened TSDBs; if this is not the case and you get this error, you must close the redundant TSDB directories using the close_tsdb command.

- -silent

  An optional switch that suppress the warning that is normally generated if the specified directory_path is already opened.

Examples

The following example opens three TSDB directories with one command invocation.

```
> open_tsdb [list ${tsdb_home}/corea.tsdb \
                  ${tsdb_home}/coreb.tsdb \
                  ${tsdb_home}/corec.tsdb ]
> get_tsdb_list
/home/projects/chipa/tsdbs/corea.tsdb /home/projects/chipa/tsdbs/coreb.t
sdb /home/projects/chipa/tsdbs/corec.tsdb
```

Related Topics

close_tsdb

get_tsdb_list

set_tsdb_output_directory

get_tsdb_output_directory

open_visualizer

Context: unspecified, all contexts

Mode: all modes

Usage

open_visualizer [-Display ***window*** **|** ***-Restart***]

Description

Opens the main DFTVisualizer window.

You can also use the following commands to open DFTVisualizer and display the Debug window:

- analyze_drc_violation
- add_display_instances
- analyze_fault (Tessent FastScan and Tessent TestKompress only)

Displayed data remains persistent throughout a tool session.

Arguments

- -Display ***window***

  An optional switch and literal pair that specifies which windows display when DFTVisualizer opens. Valid literals include:

  DEBug — opens the Debug window.

  DESign — opens the Design window.

  Browser — opens the Browser window which contains the Hierarchy, Library, and Clocks tabs.

  SIgnals — opens the Signals window.

  DAta — opens the Data window.

  Wave — opens the Wave window.

  SEarch — opens the Global Search window.

  Format — opens the Format Guide window.

  TRanscript — opens the Transcript window.

  TAskmgr — opens the Task Manager window.

  TExt_editor — opens the Text Editor window.

  TESt_structures — opens the Test Structures window.

- -Restart

  An optional -Restart switch that enables you to restart DFTVisualizer following an application error in DFTVisualizer without restarting the entire tool. This eliminates the need to exit and re-invoke the parent application.

  This option closes the current DFTVisualizer window and opens a new DFTVisualizer session displaying either the default windows normally opened at invocation or those windows specified to be open in your saved preferences. This option does not restore any of the tool data contained in DFTVisualizer windows prior to the error.

Tip: If you execute the same set of commands that resulted in the application locking up, the same error will occur and the application will lock up again. You can use the -Restart option and then avoid the same error a second time by following a different set of steps or by skipping the step that resulted in the error.

Examples

The following example opens DFTVisualizer and displays the Debug, Design, and Transcript windows.

```
open_visualizer -display debug design transcript
```

Related Commands

close_visualizer
set_visualizer_preferences
write_window_contents

order_patterns

Context: dft -edt, patterns -scan

Mode: analysis

Usage

order_patterns [*number_of_passes* | Auto ***percentage***]
[-Init_order Normal | REverse | RAndom] [-Simulate]
[{-CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average]
| -SHIft_power **LOAD | RESPonse**}[-INSTance ***instance_name***
| -MODUle ***module_name***] [-ASCending | -DEScending]]

Description

Reorders the internal test pattern set as specified.

Test patterns can be rearranged to detect the most faults first, which is useful for:

- Truncating the test pattern set in order to fit it in the ATE memory
- Detecting failing chips as early as possible during testing.

Multiple load patterns are ranked according to the average number of faults detected per scan load, ensuring that on average the patterns that detect the most faults per scan load are first. This command supports stuck-at and transition fault models, as well as timing-aware ATPG. This command does not support MacroTest or parametric patterns.

The order_patterns command does not remove ineffective patterns. If you want to remove ineffective patterns, use the compress_patterns command.

This is a multiprocessing command. For more information about multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

Note

Using the order_patterns command turns off pattern classification until you issue a "set_pattern_classification On" command.

Arguments

- *number_of_passes*

  An optional integer that specifies the number of passes to make through the data when reordering the pattern set. By default, one pass is made.

- -Auto *percentage*

  An optional switch and integer pair that reorders the pattern set iteratively until further reordering produces negligible improvement in the number of patterns needed to achieve a given fault coverage benchmark. The tool determines the given fault coverage benchmark by subtracting *percentage* from the total fault coverage (FC) achieved using all the patterns

in the internal pattern set. The *percentage* argument must be a non-negative number less than or equal to FC.

Reordering stops when either of the following conditions exist:

- The tool completes a maximum of 100 iterations.
- The number of patterns that is needed to achieve the given fault coverage benchmark decreases by less than 1% from one iteration to the next. This stop point can be stated mathematically as:

$(P_{FCb})_{previous_pass} - (P_{FCb})_{current_pass} < (P_{FCb})_{previous_pass} \times .01$

where: P_{FCb} = number of patterns needed to achieve the fault coverage benchmark
FCb = FC - *percentage* = fault coverage benchmark
FC = total fault coverage achieved using all the patterns in the pattern set

- -Init_order Normal | REverse | Random

  An optional switch and literal pair that specifies the initial pattern ordering to use as a starting point for pattern reordering. The literal choices are as follows:

  Normal — A literal that specifies the order of the current internal pattern set as the initial pattern order.

  REverse — A literal that specifies to use the reverse order of the current internal pattern set as the initial pattern order.

  Random — A literal that specifies to randomize the order of the current internal pattern set, and use the random order as the initial pattern order.

- -Simulate

  An optional switch that fault simulates the test patterns after reordering.

- -CAPture_power [ST_Peak | ST_Average | WSA_Peak | WSA_Average]

  Optional switch and literal pair that orders test patterns based on switching activity during capture. Literal options include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  WSA_Average — Average of weighted switching activity.

- -INSTance ***instance_pathname***

  An optional switch and string that orders test patterns based on switching activity associated with a specified instance. The instance_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -MODule ***module_name***

  An optional switch and string that orders test patterns based on switching activity associated with a specified module. The module_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -SHIft_power {**LOAD** | **RESPonse**}

  Optional switch and literal that orders test patterns based on switching activity during shift. Literal options include:

  LOAD — based test pattern loading.

  RESPonse — based on circuit response.

- -ASCending | -DEScending

  Optional switch that orders test patterns with respect to the amount of switching activity. By default test patterns are ordered in an ascending order.

Example 1

Total fault coverage is 98%, and there are 2,500 patterns in memory. The following example reorders patterns in the internal pattern set until the difference between the number of patterns that is needed to achieve 93% (98 - 5) fault coverage in the current pass and the previous pass is less than 1% of the number of patterns in the previous pass:

order_patterns auto 5

Example 2

The following example creates test patterns, and then reorders them based on the average weighted switching activity in capture cycles:

create_patterns
order_patterns -capture_power wsa_avg ascending

```
//  Warning: Pattern classification is turned off.
```

Related Commands

add_processors
report_patterns
compress_patterns
set_pattern_buffer
set_pattern_classification
set_pattern_filtering

printenv

Context: unspecified, all contexts

Mode: all modes

Usage

printenv

Description

Prints out the values of the Linux variables in the environment.

The printenv command allows the Linux printenv command to be available as a common DFT command, for convenience in displaying Linux environment variables. Linux environment variables are automatically available as variable references within the tool.

Examples

The following example prints out the values of the Linux variables in the environment:

```
printenv
```

process_dft_specification

Context: dft

Mode: setup, analysis

Usage

process_dft_specification [*id*] [-no_insertion [-design_name *design_name*]] [-validate_only] [-transcript_insertion_commands]

Description

Validates and processes the content contained in a DftSpecification wrapper.

If you specify the -validate_only option, this command only validates the Dft specification. If you specify the -no_insertion option, the command validates and generates the Dft components. When used without either of these two options, the command validates the specification, generates the Dft components such as the RTL and the ICL description, and inserts them into the design. The insertion can be done at the netlist or the RTL level depending on the set_context -no_rtl versus -rtl option that was specified when setting the context.

If the DftSpecification being processed has a HostScanInterface/Interface consisting of only ports on the current design and the -no_insertion option is not specified, the add_icl_scan_interfaces and the set_icl_scan_interface_ports commands are issued during the insertion stage such that if you perform ICL extraction after the insertion, the ScanInterface will be named using the *id* of the HostScanInterface contained in the DftSpecification.

The data created by this command is stored inside the directory returned by the get_tsdb_output_directory command which defaults to *./tsdb_outdir* unless it has been changed by the set_tsdb_output_directory command.

process_dft_specification.post_insertion Procedure

When performing the insertion (i.e. when neither the -validate_only nor -no_insertion switch is specified), the modified design is written out inside the "*tsdb_output_directory/dft_inserted_designs*" directory. If you need to perform custom editing between the insertion of the Dft components and the saving of the design, you can define a Tcl proc with the name "process_dft_specification.post_insertion" and it will automatically be called after all Dft components have been inserted and before the write_design command is invoked.

To develop this type of procedure, execute all of your custom editing commands interactively in Tessent Shell in the insertion mode. When you are finished, create the Tcl proc with those commands in it; the Tcl proc will then be called when rerunning process_dft_specification. If you need to instantiate new modules inside the design as part of this Tcl proc, you must read them in using read_verilog or read_vhdl in setup mode prior to invoking process_dft_specification.

The process_dft_specification.post_insertion procedure must have two arguments as shown in the following example. The first argument holds the configuration element object that is currently being processed. The second argument holds a list of option value pairs corresponding to the options that were used when the process_dft_specification command was called.

```
proc process_dft_specification.post_insertion { root_wrapper args } {

  # "$root_wrapper" is the config_element object being processed.
  # "get_config_value $root_wrapper -id 0" returns the design_name
  # "get_config_value $root_wrapper -id 1" returns the id

  # "$args" is a list of option value pairs that reflects the options used
  # when the process_dft_specification command was called.

  array set extra_args $args

  # $extra_args(-instrument_prefix) returns the value of that option

}
```

Licensing

This command normally requires an IJTAG product license to run. There is one exception to this requirement. If the design contains one or more instances of an EDT/LBIST controller, the TestKompress/LogicBist licenses are also allowed to run this command. In this case, the tool checks the availability of the license in the following order: IJTAG, TestKompress, LogicBist. If none are currently available, the tool queues up for the first license that you own.

You can refer to the set_license_queue_timeout and get_license_queue_timeout commands for more information about the license queuing feature.

Arguments

- *id*

  An optional string that specifies the DftSpecification wrapper that is to be processed. As described in the DftSpecification wrapper syntax, the wrapper is identified by the two strings *design_name* and *id*. If you only have one DftSpecification wrapper present in memory that matches DftSpecification(*design_name*,*), you can omit the id from the process_dft_specification command. However, if several DftSpecification wrappers matching DftSpecification(*design_name*,*) exist in memory, you must specify the appropriate id when executing the process_dft_specification command. Refer to the following -design_name option to understand how the *design_name* value is obtained.

- -no_insertion

  An optional switch that specifies to not perform the insertion of the Dft components into the design. When this option is specified, no design is needed because no insertion is performed. Some validations, such as verifying that the specified parent_instance properties in the Sib, ScanMux and Tdr wrappers actually exist in the design, are only performed when this switch is omitted. If you execute the process_dft_specification command without this option and the set_current_design command has not yet been issued, an error is generated.

- -design_name *design_name*

  An optional switch and string pair that specifies the *design_name* value used to identify the DftSpecification wrapper when the -no_insertion option is specified. The -design_name value defaults to the current design if it has been set with the set_current_design command.

If the current design has not been set, it is extracted from the DftSpecification wrapper existing in memory.

When the id is specified, if exactly one wrapper matches “DftSpecification (*,<id>)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified. When the id is not specified, if exactly one wrapper matches “DftSpecification (*,*)” in memory, the design_name is extracted from the design_name parameter of the DftSpecification wrapper; if more than one wrapper matches, the -design_name must be specified.

- -validate_only

  An optional switch that specifies to only perform the validation of the DftSpecification and to skip Dft element generation and insertion. Using the -no_insertion switch with the -validate_only option disables a few checks such as verifying that the specified parent_instance properties in the Sib, ScanMux and Tdr wrappers actually exist in the design.

- -transcript_insertion_commands

  An optional switch that specifies to enable the transcription of the insertion commands as they are being performed. This option can generate a large number of transcripted commands but you may find it useful to see the instantiation and connection commands created by the process_dft_specification command.

Example

The following example reads a design and a DftSpecification file, and then processes the DftSpecification. The context is changed to the patterns -ijtag context to extract the ICL of the design and generate simulation patterns. This example uses the DftSpecification shown in the example of the ScanMux wrapper documentation. Notice the use of the iNote “tessent_pragma verilog_insert” to inject native Verilog commands into the testbench at specific times to directly monitor the states of internal signals.

```
#-------------- Insert Ijtag Network --------------------
set_context dft -no_rtl
set_tsdb_output_directory scanMux_ex1_outdir
read_verilog ../data/MyCore.v
set_current_design MyCore
read_config_data ../data/ScanMux_ex1.dft_spec
process_dft_specification

#---------------- Extract ICL ---------------------------
set_system_mode setup
set_context patterns -ijtag
set_system_mode analysis
write_icl -out ScanMux_ex1.icl

#---------------- create Simulation Patterns  -----------
set dut "MyCore_ScanMux_ex1_v_ctl.MyCore_inst"
open_pattern_set p1

iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b0 and end_in_pause"
```

```
iWrite tessent_rtl_tdr_orange_inst.tdr 0b0
iApply -end_in_pause

iNote "iRead tessent_rtl_tdr_orange_inst.tdr 0b0"
iNote "iWrite tessent_rtl_tdr_orange_inst.tdr 0b1 and end_in_pause"
iRead tessent_rtl_tdr_orange_inst.tdr 0b0
iWrite tessent_rtl_tdr_orange_inst.tdr 0b1
iApply -end_in_pause

iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
  measured tessent_rtl_tdr_orange_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_orange_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
 measured tessent_rtl_tdr_green_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_green_inst.ijtag_sel);"
iNote "tessent_pragma verilog_insert \$display(\$time, \"ns:\
 measured tessent_rtl_tdr_red_inst.ijtag_sel = %b\",\
 ${dut}.tessent_rtl_tdr_red_inst.ijtag_sel);"

close_pattern_set

write_patterns ScanMux_ex1.v -ver
```

Related Commands

display_specification

read_config_data

get_tsdb_output_directory

set_tsdb_output_directory

range_collection

Context: unspecified, all contexts

Mode: all modes

Usage

range_collection ***collection first last***

Description

Returns one or more adjacent elements from a collection and creates a new collection.

The returned collection consists of elements first through last, inclusive. If first is greater than last, this command returns an empty collection.

The base collection remains unchanged.

Arguments

- ***collection***

  A required string that specifies the collection to be searched.

- ***first***

  A required integer that specifies the first index into the collection. Allowed values are from 0 to *sizeof_collection* - 1.

- ***last***

  A required integer that specifies the ending index into the collection. Allowed values are from 0 to *sizeof_collection* - 1. You can also specify the value of the last argument relative to the end of the collection by using "end-<integer>".

Example

The following example returns a collection with the last two elements from $mycoll.

```
range_collection $mycoll end-1 end
```

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
is_collection
remove_from_collection
sizeof_collection
sort_collection

read_cell_library

Context: unspecified, all contexts

Mode: setup

Usage

read_cell_library ***lib_path*** ... [-force]

Description

Loads one or more cell libraries into the tool.

This command provides functionality equivalent to the -Library tool invocation switch.

You can issue this command multiple times. In case of parsing errors, the tool clears all the library data and returns to Setup mode. After the erroneous file has been corrected and reloaded, the tool reparses all files that were successfully parsed previously.

Arguments

- ***lib_path***

  A required and repeatable string that specifies the names of the files containing the ATPG library descriptions for cell models in your design. You can use wildcards to specify multiple file names.

- -force

  An optional switch to suppress the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a library using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), executing the read_cell_library command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you execute the command with the -force switch to indicate that it is acceptable to delete the setup data.

Example

The following example loads two cell libraries:

```
read_cell_library lib1 lib2
```

Related Commands

delete_cell_library
set_cell_library_options
write_cell_library

read_config_data

Context: unspecified, dft, patterns

Mode: all

Usage

read_config_data {*filename_list* | -from_string *string*}
[-in_wrapper *parent_wrapper*]
[-before *before* | -after *after* | -first | -last]
[-replace]

Description

Reads the content of configuration data files or a string into the Tessent Shell environment.

The configuration data file format is currently used for two released features but it will be used to describe many more specifications in the future.

The first usage is to load a dft specification and a defaults specification containing the syntax described in which is used by the and commands.

The second usage is to load the test setup specifications described in "The Configuration Data File for Tessent SiliconInsight Desktop" to allow Tessent SiliconInsight to communicate via hardware adaptors or the SimDUT tool with the design under test (DUT). In this application, Tessent Shell uses the settings in the configuration data file to launch the SiliconInsight Desktop (SID) tester when the launch_sid_tester command is invoked.

Arguments

- *filename_list*

  An optional string that specifies the name of a file or a tcl list of file names. The names can include wildcards (the * character) to allow matching multiple file names with a single pattern.

- -from_string *string*

  An optional option value pair that specifies a string to parse. the string can be contained in a tcl variable or typed between curly brackets as shown in Example 3

- -in_wrapper *parent_wrapper*

  An option value pair that specifies that the read in configuration elements are to be loaded inside a specific wrapper. The *parent_wrapper* can be the complete name of a wrapper relative to the root of the partition or a collection containing a single wrapper element as returned by or " -object". When the -in_wrapper is not specified, the destination parent wrapper is assumed to be the root of the partition unless the -before or -after option is specified.

  If the specified root wrapper is a wrapper of type CSV, you can load a CSV file generated out of Excel™ where the line terminator is the carriage return instead of a semicolon.

- -before *before*

  An optional option value pair that specifies a configuration element before which to the new configuration elements are to be loaded. When the -before option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

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

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

- -after *after*

  An optional option value pair that specifies a configuration element after which the new configuration elements are to be loaded. When the -after option is used with a collection containing a configuration element object, the -in_wrapper option cannot be used as the parent wrapper is extracted from the object.

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

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

- -first

  An optional option that specifies the new configuration elements are to be loaded before any other element in the parent wrapper.

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

- -last

  An optional option that specifies the new configuration elements are to be loaded after all other elements in the parent wrapper.

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

- -replace

  An optional options that suppresses the error that is normally generated when loading configuration elements having the same name as an existing element in the destination wrapper. The error only occurs if the name is defined as not repeatable in the metadata. Instead of an error, the existing element is first deleted.

Example

The following example loads the content of the mydesign.dft_spec_rtl file into the Tessent Shell environment.

read_config_data mydesign.dft_spec_rtl

Refer to "Executing an IJTAG Test Pattern" in *Tessent SiliconInsight User's Manual for Tessent Shell* for more usage scenarios.

Example 2

The following example reads a CSV file into a wrapper of type CSV.

```
add_config_element csv(1)
read_config_data data.csv -in_wrapper csv(1)
report_config_data csv(1)
```

Example 3

The following example illustrates the use of the -from_string option. The content of the string was created by the " -wrapper DftSpecification(corea,rtl)/IjtagNetwork/HostScanInterface(custom)" command. The HostScanInterface(custom) was created manually in the Configuration Data window. Because the read_config_data command is inside the callback, the custom HostScanInterface wrapper will automatically inserts itself into the newly generated DftSpecification wrapper when the is called. You could have also used DftSpecification/IjtagNetwork/HostScanInterface to get the available syntax and type it in manually.

```
proc create_dft_specification.post_create (dft_spec_name args} {
  read_config_data -in $dft_spec_name/IjtagNetwork -from_string {
    HostScanInterface(custom) {
      Sib(power_management) {
        DesignInstance(myTDR1) {
        }
        DesignInstance(myTDR2) {
        }
      }
    }
  }
}
```

Related Topics

Configuration-Based Specification	launch_sid_tester
display_specification	The Configuration Data File for Tessent SiliconInsight Desktop in the *Tessent SiliconInsight User's Manual for Tessent Shell*
process_dft_specification	The Configuration Data File for SimDUT

read_core_descriptions

Context: all contexts

Mode: setup

Usage

read_core_descriptions ***file_names***

Description

Reads the specified Tessent core description (TCD) files.

The Tessent core description file is used to map scan information from one level to the next for running ATPG at the higher level, and to retarget patterns for instances of the associated core.

Arguments

- ***file_names***

  A required, repeatable string that specifies the core description files to be read. The names of the files can contain wildcards. You can define the names of files to read into a Tcl list, and use the name of the Tcl list as the file_names argument.

Example

The "Retargeting Example" in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Commands

add_core_instances
delete_core_descriptions
delete_core_instances
report_core_descriptions
report_core_instances
write_core_description

read_cpf

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Usage

read_cpf ***filename*** [{> | >>} *file_pathname*]

Description

Loads a CPF (Common Power Format) file and removes any previously loaded power data. You can use multiple CPF files by using the native CPF “include” command.

The tools support CPF version 1.0 and 1.1.

Effects of Reading Power Data

Loading power data from a CPF or UPF file has the following effects:

- Simulation of test procedures and scan patterns take into account whether a gate is powered on during simulation.
- The tool runs the Power-Aware Rules (V Rules) when transitioning out of Setup mode.
- Scan insertion restricts scan stitching such that each scan chain includes only scan cells from one power domain.
- Test point sharing is restricted based on power information.
- The power attributes provide access to the power information. The power attributes are:

Data Model	Power Attribute
	is_power_always_on_cell is_power_isolation_cell is_power_level_shifter_cell is_power_retention_cell
, ,	power_domain_name

Power information does not affect dft or analysis in any other way. For example, when creating connections across power domains, the tools do not automatically add isolation cells or level shifter cells.

Arguments

- ***filename***

  A required string that specifies the name of the CPF file.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example loads a CPF file:

read_cpf cpf_file

Related Commands

add_faults
delete_faults
read_faults
read_upf
report_faults
report_power_data

read_failures

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

Mode: analysis

Usage

read_failures ***failure_file*** [-Pattern | -Cycle] [{> | >>} *output_file*]

read_failures ***failure_file*** [-Append]

Description

Checks the specified failure file for correct syntax and semantics and optionally appends a failure file.

By default, this command displays the first 20 pattern mismatches. Use the command to change the number of pattern mismatches that display.

You can also use the read_failures command to display additional information about failed cells within the failure file, or use it to convert a failure file to either pattern-based or cycle-based format. This command supports both, compressed and uncompressed test patterns.

Test patterns must be WGL or STIL format for cycle-based failure files.

When read_failures -append is used, then the order in which the failure files are supplied must match the order in which the multiple patterns sets have been loaded. In the event that one of the failure files is inconsistent with the corresponding test set, an error will be issued and that failure file will not be appended to the previously loaded failure files.

Note

If you omit the -append switch, then the tool overwrites the in-memory failure file.

Tessent Diagnosis performs a basic syntax check during parsing.

Arguments

- ***failure_file***

  A required string that specifies the name of the failure file to read. For failure mapping, this file is a top-level failure file generated as a result of applying top-level patterns created by scan pattern retargeting.

- -Pattern

  An optional switch that displays/writes the failure file in the pattern-based format. Use this switch to convert a cycle-based failure file to a pattern based format or use it to add location information to failed cells within a pattern-based failure file.

- -Cycle

  An optional switch that displays/writes the failure file in the cycle-based format. Use this switch to convert a pattern-based failure file to a cycle-based format or use it to add location information to failed cells within a cycle-based failure file.

- -Append

  An optional switch that appends the *failure_file* to a previously-specified failure file and saves the appended *failure_file* in memory.

  Before appending a failure file, you must issue the read_failures command without the -append switch and, subsequently, issue the read_failures command with the -append switch.

  > *output_file*

  An optional redirection operator and pathname pair used at the end of the argument list to create *or* replace the contents of *output_file*.

- >> *output_file*

  An optional redirection operator and pathname pair used at the end of the argument list to append output to the contents of *output_file*.

Example 1

The following example loads a test pattern source (*1pat.wgl*), reads in a pattern-based failure file (*padding_failures.pat*) and writes out a cycle-based failure file (*pat_cycle.fail*).

```
read_patterns 1pat.wgl
read_failures fail_files/padding_failures.pat -cycle > results/pat_cycle.fail
```

Example 2

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_failures failure_file1

read_failures failure_file2 -append
```

Example 3

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

```
read_patterns pat1

read_patterns pat2 -append

read_failures failure_file1

read_failures failure_file2 -append
```

Example 4

The following example shows the *padding_failures.pat* file.

```
chain_test
0  chain5     0   H  L
0  chain5     1   H  L
0  chain5     4   H  L
0  chain5     5   H  L
0  chain5     8   H  L
0  chain5     9   H  L
0  chain5     12  H  L
0  chain5     13  H  L
0  chain5     16  H  L
0  chain5     88  H  L
0  chain5     100 H  L
last_pattern_tested  0
scan_test
0   chain5    2   H  L
0   chain5    4   H  L
0   chain5    6   H  L
0   chain5    8   H  L
0   chain5    9   H  L
0   chain5    11  H  L
0   chain5    12  H  L
0   chain5    13  H  L
0   chain5    28  H  L
0   chain5    100 H  L
last_pattern_tested  2
```

Example 5

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

read_failures failure_file1

read_failures failure_file2 -append

Example 6

The following example shows the cycle-based failure file (*pat_cycle.fail*). The cycle-based data is written in columns from left to right: cycle id, pin name, expected value, simulated value, pattern id, chain/PO, cell number, cell path name.

```
format cycle
failures_begin
// chain test
105 dhiraj_so5 H L  // pat 0  chain chain5  cell 0   uUART/reg_MODE0_CLK
106 dhiraj_so5 H L  // pat 0  chain chain5  cell 1   uCNTR/reg_PORT_3_SAMPLE_P1_2_
109 dhiraj_so5 H L  // pat 0  chain chain5  cell 4   uCNTR/reg_PORT_3_SAMPLE_P1_5_
110 dhiraj_so5 H L  // pat 0  chain chain5  cell 5   uINTR/ix546
113 dhiraj_so5 H L  // pat 0  chain chain5  cell 8   uINTR/ix476
114 dhiraj_so5 H L  // pat 0  chain chain5  cell 9   uINTR/ix446
117 dhiraj_so5 H L  // pat 0  chain chain5  cell 12  uINTR/ix356
129 dhiraj_so5 H L  // pat 0  chain chain5  cell 24  uSTM/ix56/SFFR
 ...
// scan test
314 dhiraj_so5 H L  // pat 0  chain chain5  cell 2   uCNTR/reg_PORT_3_SAMPLE_P1_3_
316 dhiraj_so5 H L  // pat 0  chain chain5  cell 4   uCNTR/reg_PORT_3_SAMPLE_P1_5_
318 dhiraj_so5 H L  // pat 0  chain chain5  cell 6   uINTR/ix106
320 dhiraj_so5 H L  // pat 0  chain chain5  cell 8   uINTR/ix476
321 dhiraj_so5 H L  // pat 0  chain chain5  cell 9   uINTR/ix446
323 dhiraj_so5 H L  // pat 0  chain chain5  cell 11  uINTR/ix386
621 dhiraj_so5 H L  // pat 2  chain chain5  cell 101 padding_cycle
failure_buffer_limit_reached all
failures_end
```

Example 7

The following example loads a test pattern source (*1pat.wgl*), reads in a cycle-based failure file (*pat_cycle.fail*) and writes out a pattern-based failure file (*cyc_pat.fail*).

```
read_patterns 1pat.wgl
read_failures fail_files/pat_cycle.fail -pattern > results/cyc_pat.fail
```

Example 8

The following example shows the pattern-based failure file (*cyc_pat.fail*). The pattern-based failure file data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name:

```
chain_test
0 chain5   0   H  L  //  uUART/reg_MODE0_CLK
0 chain5   1   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   5   H  L  //  uINTR/ix546
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   16  H  L  //  uINTR/ix236
0 chain5   88  H  L  //  padding_cycle
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  0
scan_test
0 chain5   2   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   6   H  L  //  uINTR/ix106
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   11  H  L  //  uINTR/ix386
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   28  H  L  //  uSTM/reg_WAIT_SYNC/SFFR
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  2
```

Example 9

The following example loads a test pattern source(*1pat.wgl*), reads in a compressed pattern-based failure file (*fail_file.pat*) and writes out a compressed pattern-based failure file (*pat_2_pat*) that includes cell path names for internal scan chains corresponding to each failing EDT channel.

```
read_patterns 1pat.wgl
read_failures fail_file.pat -pat > results/pat_2_pat
```

Example 10

The following example shows the compressed pattern-based failure file (*fail_file.pat*) loaded into the tool.

```
chain test
   0  edt_channel1  2  H  L
   0  edt_channel1  3  L  H
   0  edt_channel2  0  L  H
   0  edt_channel2  4  H  L
   1  edt_channel1  0  H  L
last_pattern_tested 1
scan test
   0  /x_out[4]  H  L
   0  /x_out[2]  H  L
   0  /y_out[5]  L  H
   0  /y_out[4]  L  H
   0  edt_channel1  0  L  H
   0  edt_channel1  1  H  L
   0  edt_channel1  2  L  H
   1  /x_out[3]  H  L
   1  /x_out[2]  H  L
   1  /x_out[1]  H  L
last_pattern_tested 1
```

Example 11

The following example shows the resulting compressed pattern-based failure file (*pat_2_pat*) output by the tool. Data is written in columns from left to right: pattern_id, channel/PO_name, channel_cycle_id, expected_value, simulated_value, chain---cell_id---cell_path_name. For

compressed patterns, the location of every possible failing cell in the internal scan chains corresponding to each failing EDT channel is listed.

```
chain_test
0  edt_channel1  2  H  L   //  chain1---2---circle_i/reg_x_2_/DF1
                            //  chain2---2---circle_i/reg_f_6_
                            //  chain3---2---circle_i/reg_f_1_
0  edt_channel1  3  L  H   //  chain1---3---circle_i/reg_x_3_/DF1
                            //  chain2---3---circle_i/reg_f_7_
                            //  chain3---3---circle_i/reg_f_2_
0  edt_channel2  0  L  H   //  chain4---0---circle_i/reg_y_2_
                            //  chain5---0---circle_i/reg_x_5_/DF1
0  edt_channel2  4  H  L   //  chain4---4---circle_i/reg_y_6_
                            //  chain5---4---circle_i/reg_y_1_
1  edt_channel1  0  H  L   //  chain1---0---circle_i/reg_x_0_/DF1
                            //  chain2---0---circle_i/reg_f_4_
                            //  chain3---0---circle_i/reg_y_7_
last_pattern_tested  1
scan_test
0  /x_out[4]        H  L   //  PO
0  /x_out[2]        H  L   //  PO
0  /y_out[5]        L  H   //  PO
0  /y_out[4]        L  H   //  PO
0  edt_channel1  0  L  H   //  chain1---0---circle_i/reg_x_0_/DF1
                            //  chain2---0---circle_i/reg_f_4_
                            //  chain3---0---circle_i/reg_y_7_
0  edt_channel1  1  H  L   //  chain1---1---circle_i/reg_x_1_/DF1
                            //  chain2---1---circle_i/reg_f_5_
                            //  chain3---1---circle_i/reg_f_0_
0  edt_channel1  2  L  H   //  chain1---2---circle_i/reg_x_2_/DF1
1  /x_out[3]        H  L   //  PO
1  /x_out[2]        H  L   //  PO
1  /x_out[1]        H  L   //  PO
last_pattern_tested  1
```

Example 12

In the following command sequence example, the tool appends *failure_file2* to *failure_file1*:

read_patterns pat1

read_patterns pat2 -append

read_failures failure_file1

read_failures failure_file2 -append

Example 13

The following example loads the top-level *fail1.top* failure file.

read_failures fail1.top

Related Commands

diagnose_failures
report_failures
set_diagnosis_options
write_failures

read_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Faults Usage:
read_fault_sites ***filename***… [-Force | -Noforce]
[-INSTance *instance_name*... | -MODule *module_name*...] [{> | >>} *output_file_name*]

Bridge Faults Usage:
read_fault_sites ***filename***…
[-INSTance *instance_name*... | -MODule *module_name*...]
[-VErbose [All | Modules | Instances]] [-NAmes_ignored]
{[-NOEQ] |
{[-Filter {DISTance | PARAllel_run | WEIGHT} *operator number*]…
[-Filter TYPE {= | !=} *type_identifier*]…
[-Filter LAYER {= | !=} *string*]…
[-SElect *number*[%]]…}} [{> | >>} *output_file_name*]

UDFM Faults Usage:
read_fault_sites ***filename*** [-VErbose]
[-INSTance *instance_name*... | -MODule *module_name*...]

Description

Loads the path or bridge definitions or user-defined fault models (UDFM) contained in the specified ASCII file into the current tool session.

Bridge Fault Specifics

A bridge fault definition file or output file from the Calibre query server can be loaded with this command. If a syntax error is encountered, the tool issues an error message and stops loading the bridge entries from the file. If a non-syntax error occurs, an error displays but the file continues loading.

After bridge loading completes, the tool displays a summary of all errors and warnings encountered during the load. The summary lists how many bridges could not be mapped and why. For example, a bridge cannot be mapped if it relates to a location within a library cell.

Note

If you define one or more filters using the -Filter switch, the command will load only those bridges that satisfy all specified filter criteria.

For more information on bridge faults and the bridge fault definition file, refer to "Net Pair Identification with Calibre for Bridge Fault Test Patterns" in the *Tessent Scan and ATPG User's Manual*.

Path Delay Specifics

Before you use the tool to create ATPG patterns to detect path delay faults, first place the paths into a properly formatted file. Then use this command to load the file, and the add_faults or read_faults command to add the faults on the specified paths. You must also specify the path delay fault type with the set_fault_type command.

If you return to Setup mode with faults in the path delay fault list, the tool deletes the faults in that list and issues a warning message.

You can use multiple read_fault_sites commands and the results are additive. However, if you do not use the -Force switch, and one of the following conditions is not met when the tool reads in the path data, the tool will issue an error message and terminate the command:

- The path name must not be the same as the name of an existing path.
- The first pin for a path must be a valid launch point. A valid launch point is a primary output of a scan cell state element, or a non-scan state element that satisfies the C1 clock rule.

  If the path includes a clock or state element D-input pin, you must include the state element name in the path (or use the -Force switch). Fail to do so, and the tool will not resolve the path and report an error.
- The last pin for a path must be a valid capture point or a clock input of a scan cell. A valid capture point is a primary output or a data input of a cell that can capture data, whether scanned or not. Examples of invalid capture points would be a latch that is transparent or a memory element that cannot capture data due to a DRC violation, pin constraint, or cell constraint.
- Each pin must have unambiguous fan-in connectivity to the preceding pin, which must not tie to a constant logic value. If the pin fails to have a valid connection with the preceding pin, the tool generates an error and terminates the read_fault_sites command. However, if there is ambiguity in the connectivity, the tool selects a path between the pin and the preceding pin and generates a warning message. You can display the gates in the complete path using the Report Path command.
- Paths cannot propagate through RAM gates, ROM gates, or transparent latches.
- Paths cannot have edge ambiguity during any point in the path. An edge that propagates through XOR gates or the select lines of MUX gates can result in either a rising or falling edge at the gate outputs. You can use inversion parity to avoid edge ambiguity. If this check fails, the tool generates a warning, and you can assume that an edge on the pin is not inverted relative to the preceding pin.
- The condition statements in the path definition file must occur before the first pin statement and before the tool checks for valid pin names and values. The tool does not use the conditions to resolve edge or path ambiguities.

For more information on path delay faults and the path definition file, refer to "Creating a Path Delay Test Set" in the *Tessent Scan and ATPG User's Manual*.

UDFM Specifics

You use this command to load a UDFM definition file. If the tool encounters an error during loading, the tool stops the loading operation and displays a message. When the UDFM file is successfully loaded, the tool displays a summary similar to the following:

```
// UDFM definitions read: Total=1532, Cells=92, Modules=0, Instances=0,
//       No Match=476
```

The summary lists how many UDFM definitions were read from the file and how many of them have been added to the list of fault sites. In the above example, 1532 UDFM definitions are read from the file and 92 cells are added to the list of fault sites. The last value gives you the number of the UDFM definitions that did not match a cell or module in the design. You can use the -Verbose switch to report more detailed warning messages about the UDFM definitions that were not added to the list of fault sites.

For more information about UDFM, refer to "Understanding Test Types and Fault Models" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ***filename***

  A required repeatable string that specifies the pathname of the path or bridge definition file or the UDFM file to load. This file is an ASCII file that defines the path delay or bridge fault sites or user-defined fault models for ATPG.

- -Force

  An optional switch that specifies continued path reading after the first occurrence of an invalid path. If you do not specify this switch and the command encounters an invalid path, the command generates an error and terminates.

- -Noforce

  An optional switch that specifies for the tool to stop path reading after the first occurrence of an invalid path. This is the default.

- -INSTance *instance_name*...

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

- -MODule *module_name*...

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose

actual module name matches the regular expression. The fault file will be loaded for each instance.

- -VErbose

  An optional switch that enables the display of parsing errors and warnings while the path, bridge, or UDFM ASCII file is loaded.

  For bridge entries only, this switch displays a summary of all errors and warnings once bridge entries are loaded. You can specify how these errors and warnings display with one of the following literals:

  All — displays all errors/warnings during loading and a summary report after loading.

  Modules — displays the number of bridges along with the names of the related modules that could not be mapped.

  Instances — displays the number of bridges along with the names of the related modules and instances that could not be mapped.

  By default, only the summary report displays at the conclusion of the load operation.

- -NAmes_ignored

  An optional switch that directs the tool to ignore all bridge names while loading. Use this option to load files having duplicate bridge names in the bridge definition or Calibre query server file, or to save memory.

- -NOEQ

  An optional switch that prevents the loading of equivalent fault sites. This has the advantage of producing the smallest peak memory demand, as well as the smallest final memory demand, for loading.

- -Filter {{DISTance | PARAllel_run | WEIGHT} *operator number*}

  An optional repeatable switch, literal, operator and number quadruplet that filters which bridge entries are loaded based on a specified physical attribute of the bridge. A single blank space is required between each member of the quadruplet. The *number* must be a nonnegative number that applies to the specified attribute in the bridge entry. Attributes include:

  DISTANCE — Real number that specifies the distance attribute in the bridge entry.

  PARALLEL_run — Real number that specifies the parallel_run attribute in the bridge entry.

  WEIGHT — Real number that specifies the weight attribute in the bridge entry.

  The *operator* must be one of the following:

  < less than

  <= less than or equal to

  = equal to

  != not equal to

>= greater than or equal to

> greater than

Due to rounding, it is not advisable to use “=” or “!=” for floating point numbers.

Note

The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter TYPE {= | !=} *type_identifier*

  An optional repeatable switch, literal, operator and literal quadruplet that filters which bridge entries are loaded based on the bridge type. A single blank space is required between each member of the quadruplet. Select one of the following literals as the *type_identifier*:

  S2S — Side-to-side

  SW2S — Side-Wide-to-Side: same as S2S but at least one of the two signal lines is a wide metal line.

  S2SOW — Side-to-Side-Over-Wide: same as S2S, but the bridge is located over a wide piece of metal in a layer below.

  C2C — Corner-to-Corner

  V2V — Via-to-Via: an S2S for vias.

  VC2VC — Via-Corner-to-Via-Corner

  EOL — End-of-Line: the end head of a line faces another metal line.

Note

The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -Filter LAYER {= | !=} *string*

  An optional repeatable switch, literal, operator and string quadruplet that filters which bridge entries are loaded based on the layer attribute of the bridge. A single blank space is required between each member of the quadruplet. The *string* must be the string value that specifies the layer attribute in the bridge entry.

Note

The tool uses the information you provide with the -Filter switch only to determine a subset of bridge entries to load; the values are not used in any other way.

- -SElect *integer*[%]

  An optional repeatable switch and positive integer pair that specifies to keep the *integer* best bridges of each bridge type. The tool determines the best based on the weight of each bridge after loading all fault locations in all files specified in the command line.

If you include the optional “%” character after the *integer*, it is interpreted as a percentage instead of an absolute number. For example, “-select 20%” directs the tool to retain the best 20% of all bridges after loading, whereas “-select 20” means to keep the twenty best bridges.

Note

A weight attribute must be defined for each bridge in the file for this option to be meaningful. If a weight is not defined for a bridge, it is loaded regardless of this switch.

- > | >> *output_file_name*

  An optional switch that writes the parsing errors, warnings and summary report to the specified file. Use this switch at the end of the command line. Options include:

  > Creates a new or overwrites an existing file.

  >> Appends the output to an existing file.

Example 1

The following path delay example sets up the tool to perform ATPG for the specified path delay faults in the path definition file, */user/design/pathfile*.

First, you must set the fault type and read in the paths:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH “path0” =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

report_faults -all

```
type  code  name_of_path
----  ----  --------------------
//  Warning: No faults reported.
```

Next, add the faults on the paths contained in that file:

add_faults -all
report_faults -all

```
type  code  name_of_path
----  ----  --------------------
  0    UC   path0
  1    UC   path0
```

Now you are ready to perform ATPG on the path delay faults:

create_patterns

The following is an example of a path definition file that contains one path:

```
PATH “path0” =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge fault example sets up the tool to perform ATPG for the bridge fault sites specified in the bridge fault definition file, */user/design/bridgefile*.

First, you must set the fault type and load the bridge sites from the bridge fault definition file:

set_fault_type bridge
read_fault_sites /user/design/bridgefile

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U0.BRIDGE";
}
```

Next, add_faults on the sites in the fault site list:

add_faults –all
report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U7.BRIDGE";
```

```
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {UC, UC, UC, UC};
  NAME = "U0.BRIDGE";
}
```

Now you are ready to perform ATPG on the bridge faults:

create_patterns

Example 3

The following is an example of a bridge fault definition file that contains one bridge site:

```
VERSION 1.1

BRIDGE {
  NET1="G173";
  NET2="G1296";
  NAME="My_layout.BRIDGE_TYPE1";
  DISTANCE=4.93827e-06 um;
  PARALLEL_run=6.075e-05 um;
  WEIGHT=95
  X_COORDINATE=152500;
  Y_COORDINATE=739500;
  LAYER="metal1";
  TYPE="S2S";
}
```

Example 4

The following example filters bridge entries based on the parallel_run attribute and loads only the bridge entries whose parallel_run attribute is between 7.5e-05 and 8e-05 um into the current fault definition list.

delete_fault_sites -all
read_fault_sites /user/design/bridgefile -filter parallel_run > 7.5e-05 -filter parallel_run < 8e-05

```
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  2 bridge entries were filtered out.
//  1 bridge entry was added to the fault site list.
```

Notice the tool loads only the bridge entries that meet the specified filter criteria, but does not otherwise utilize the attributes. To see the attributes, include the -Attribute switch when you report the fault sites.

Example 5

The following example loads all bridge entries from the *z_bridge_definitions* file and writes loading warnings and errors to the *loading_errors* file:

```
read_fault_sites z_bridge_definitions > loading_errors
//  ... writing to file loading_errors
```

Example 6

In the following example, remember that the read_fault_sites does not actually add the fault to be targeted by ATPG, but only creates a named fault for referencing later. However, will automatically add the loaded faults when issued.

run 1: You create faults via physical extraction for module alu. You complete the run and are satisfied with the block coverage.

```
read_fault_sites alu_bridges_file
create_patterns
```

run 2a: You then embed the alu design from run 1 in a large design as instance alu1. You want to reuse the fault file used in run 1 to define the faults within that particular instance of the new larger design.

```
read_fault_sites alu_bridges_file -instance alu1
```

You want all bridges loaded to be targeted, except for the one in the fault file named *bridge7*. Bridge fault sites must have names in the file and you cannot have used the -names_ignored switch, which throws away the names to save internal storage.

```
delete fault site alu1/bridge7 -name
create_patterns # automatically adds all nondeleted, loaded faults ...
```

run 2b: You embed the alu design 10 times as a different instance and want to reuse the fault file from run 1 to define the faults in each of the 10 instances of module alu.

```
read_fault_sites alu_bridges_file -module alu
```

run 2c: You embed the alu design as inst1 and inst2; because no other instances begin with “nest”, you can use the * wildcard as shown:

```
read_fault_sites alu_bridges_file  -inst inst*
```

In all of these runs, you can issue after loading and deleting fault sites to obtain the desired set. The create_patterns command automatically adds faults at every loaded, undeleted fault site for ATPG to target.

Example 7

The following example loads two UDFM files into the current tool session and displays parsing errors and warnings while loading the second file:

read_fault_sites /home/design/c090_std/c090_std.udfm
read_fault_sites /home/design/c090_io/c090_io.udfm -verbose

```
//  Warning: UDFM - File c090_io.udfm, line 31 - Cell definition u_and2 does
not exist in the circuit.
```

Related Commands

add_faults
delete_fault_sites
report_fault_sites
set_fault_type
write_fault_sites

read_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay/Stuck/Transition/Toggle/Iddq Faults Usage:

read_faults ***filename***
{[-INSTance *instance_name*... | -MODule *module_name*...] [-VERBose] [-GRaybox]
[-DETection {Append | Replace}] }
{[-ALLOW_fault_sampling] | [-RETain] | [-PROtect [-Only] [-Force]] | [-Delete] |
[-DELETE_Equivalent]}
| {[-Merge [-NO_Warnings]]}

Bridge Faults Usage:

read_faults ***filename*** [-RETain]
{{[-INSTance *instance_name*... | -MODule *module_name*...]}
[-GRaybox] } [-DETection {Append | Replace}]

UDFM Faults Usage:

read_faults ***filename*** [-INSTance *instance_name*... | -MODule *module_name*...]
[-VERBose] [-GRaybox][-RETain] [-Force]

Power-Aware Options (applicable only after you have loaded CPF/UPF power data):

[-POWer_check {ON | Off}]

Description

Updates the current fault list with the faults contained in the specified fault file.

The read_faults command affects the current fault population by either adding or removing faults that you specify in an external fault file. Because you must identify the faults before performing ATPG or Fault simulation, this command is useful when you have a large number of faults to identify. The power-aware option loads faults based on the active power mode.

The read_faults command supports the MTFI (Mentor Tessent Fault Information) format. For more information about MTFI, refer to the *Tessent Scan and ATPG User's Manual*.

Path Delay/Stuck/Transition/Toggle/Iddq Faults Specifics

By default, if you do not specify the -Retain switch, the tool initially places all the faults in the fault file into the uncontrolled (UC) fault class. It then analyzes and reclassifies them appropriately.

Note

The filename specified cannot have fault information lines with comments appended to the end of the lines or fault information lines greater than six columns. If it does, the tool will not recognize the line properly and will not add the fault on that line to the fault list.

The format of the fault file data can be either in a three-, four-, or six-column standard format. Regardless of the format, the read_faults command uses only the information in the first, second, and last columns. You may also place comment lines in the file by starting the line with a double slash (//); the tool ignores comment lines. The file follows the format illustrated below:

```
stuck fault_class (cellname) (netname) (ignore) pin_pathname
```

- **stuck** — The first column must be the stuck-at value.
- **fault_class** — The second column must be the fault class value, but only if you use the -Retain option. For detailed information on fault classes, refer to the "Fault Classes" section in the *Tessent Scan and ATPG User's Manual*.
- **cellname** — The third column is the cell name enclosed in parenthesis. When present, this column indicates the type of cell in which the fault resides.
- **netname** — The fourth column is the net name enclosed in parenthesis. When present, this column indicates the net in which the fault resides.
- **ignore** — The fifth column is ignored by both tools.
- **pin_pathname** — The last column must be the pin pathname.

Here is an example of a short fault file that provides just the data the tool uses (as described in the first, second, and last bullets above):

```
// Short fault file example
1 UC /u510/u17/A0
1 UC /u510/u17/Y
0 UU /u510/u17/A1
1 UU /u510/u17/A1
```

When you issue this command, the tool discards all patterns in the current test pattern set unless you issue the "-protect -only" option. See the description of the -Only switch for more information.

Note

When black boxes are present in the design, the read_faults command is unable to load the faults that are inside the black boxes. Instead, the tool discards those faults and issues warning messages. The result is fewer faults loaded into the tool.

Arguments

- ***filename***

  A required string that specifies the name of the ASCII file containing the fault list to load. The command automatically detects when filename is an MTFI file. You can enter only one string.

- -INSTance *instance_name*...

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by the *instance_name* string for the reuse of a fault file created prior to the

instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *instance_name* argument, and the wildcard is expanded to all legal instance names. The fault file is loaded for each instance.

- -MODule *module_name...*

  An optional switch and repeatable string that causes each fault site in *filename* to be prepended by *module_name* string for reuse of a fault file created prior to the instances being embedded in the current larger design. This switch must occur after the *filename* argument. You can use the asterisk (*) and question mark (?) wildcards to specify the *module_name* argument, and the wildcard is expanded to all legal instance names whose actual module name matches the regular expression. The fault file will be loaded for each instance.

- -VERBose

  An optional switch that applies only to *-Instance* and *-Module*. Prior to being embedded, top-level IO pins on an instance are faultable sites; after being embedded, they no longer are fault sites. In this case, a summary message states that these faults cannot be loaded due to no longer being a faultable site. If the -Verbose switch is issued, an individual message for each unloadable fault is also issued. By default, this switch is disabled.

- -GRaybox

  An optional switch that causes the instances specified with the -Instance or -Module switch to be treated as gray boxes. When you use the -Graybox switch, you must use either the -Instance or -Module switch.

  When the tool can translate the loading faults to an existing net, the -Graybox switch merges the faults to the internal fault list. When the tool cannot translate a loading fault to an existing net, the tool designates that fault as unlisted and retains its detection status.

  Note that this option does not support reading multiple fault lists for the same instance or module. If your intent is to merge multiple fault lists for a given core, then you must perform the merge during the core-level ATPG run using the “read_faults -merge” command. Note also that you can only merge fault lists of a single fault type. You cannot, for example, merge fault lists for stuck and transition at the core level.

  For more information about using the -Graybox switch and hierarchical fault accounting, refer to “Support for Hierarchical Fault Accounting” in the *Tessent Scan and ATPG User’s Manual*.

- -DETection {Append | Replace }

  An optional switch and literal operand that specifies the operation on the n-detection number of faults in the FaultList data block of the loading MTFI file.

  The Append literal appends the n-detection number stored in the external MTFI file to the n-detection number of the corresponding fault. The append operation caps at the detection limit set by the command. When you use either the -Merge or -Graybox switch, the

"-Detection Append" switch automatically applies. When you use the "-Detection Append" switch, you must also include either the -Merge or -Graybox switch.

The Replace literal replaces the n-detection number of the fault in the internal fault list with the n-detection value of the corresponding fault loaded from the MTFI file. When you use the "-Detection Replace" switch, you must also use the -Retain or -Restore switch.

- -ALLOW_fault_sampling

  An optional switch enables fault sampling when loading faults.

  By default, the read_faults command ignores fault sampling rate data unless you specify this switch. If the fault sampling rate is other than 100%, and you do not specify the -ALLOW_fault_sampling switch, then the tool resets the sampling rate; for subsequent commands, you must re-specify the sampling rate.

Note

The behavior for the -Delete or -DELETE_equiv switches remain unchanged irrespective of whether the you use this switch or not.

- -RETain

  An optional switch that specifies for the tool to retain the fault class of each fault in the fault list.

- -PROtect

  An optional switch that specifies to load all faults from the external fault list file and to protect the detected faults from manipulation by subsequent tool commands.

  The tool assigns the protected faults to class DI. Use this switch when you want the tool to generate tests only for undetected faults (stuck-at, toggle, transition, or IDDQ), yet determine coverage statistics based on all detected faults. This switch does not apply to bridge faults.

Note

Any fault in the tool's current internal fault list that is identical to a protected fault being read in also becomes protected and the tool removes it from the fault list.

Caution

If you use -Protect to turn on fault protection, be sure you no longer need the protection before issuing a subsequent "set_fault_type" command. The command erases all fault information, even the information about protected faults.

-Only

An optional switch that, together with -Protect, specifies to load only the detected faults from the external fault list file and then protect them.

In contrast to other read_faults options, "-protect -only" does not delete the internal pattern set. This is because the option adds and protects faults for which it assumes a

pattern set exists elsewhere (in an external pattern file, for example). It does reset the fault classes, however, because the loaded protected faults also protect any equivalent fault in memory and this will change the coverages. Therefore, be sure to perform a fault simulation of the current pattern set

```
set_system_mode analysis
simulate_patterns
```

-Force

An optional switch that, together with -Protect, specifies for the tool to protect UC faults as well as detected faults from manipulation by subsequent tool commands.

- -Delete

  An optional switch that specifies for the tool to remove all the *filename* faults except the protected faults from the current fault population. To delete protected faults, you must use the command.

- -DELETE_Equivalent

  An optional switch that specifies for the tool to remove all the unprotected *filename* faults from the current fault population within the tool's session, and to remove all the unprotected faults that are equivalent to those in *filename*.

- -Merge

  An optional switch that merges the loaded external faults with the internal fault list. This allows you to combine the fault lists from different ATPG sessions. For example, if you partition your design and run ATPG for each partition, you can get chip-level fault statistics by merging the partition fault lists. Also, for a single design or design partition, you can run ATPG once, then you can target more undetected faults by incrementally running ATPG using different settings and merging fault lists. For more information, see Example 2.

  If a fault being loaded does not exist in the internal fault list, it will be added to the internal fault list.

  If a fault being loaded already exists in the internal fault list, the fault class of the existing fault will be updated to have with the highest priority fault class. For example, if the fault being loaded has a DI class and the matching fault in the internal list has a PT class, then the class of the fault in the internal list will be updated to DI because DI has a higher priority than PT.

  Fault classes are prioritized as follows:

  1. (Highest) Untestable Faults: RE, TI, BL
  2. Detected or Detectable Faults: DR > DS > DF > DI > PT > PU
  3. Aborted or Undetectable Faults: UO > UC > AU > UU

Note

The three untestable fault classes (RE, TI, and BL) have equal priority. It is a conflict to have a fault with more than one untestable fault class. The tool issues a warning message if a fault being loaded has a different untestable fault class than a matching fault in the internal fault list.

-NO_Warnings

An optional switch that suppresses warnings about conflicting untestable fault classes while merging fault lists. If a fault being loaded has a different untestable fault class than a matching fault in the internal fault list, the fault class defined in the internal list will be used.

- -Power_check {ON | Off}

An optional switch and literal that controls the loading of faults in a power-off domain. The default is Off. This switch is applicable only after you have loaded a CPF or UPF file.

ON — A literal that loads the faults on a power-on domain and discards the faults in a power-off domain. Use this option to calculate the overall test coverage, including the faults that are not active in the current active power mode.

Off — A literal that allows all the faults to be loaded without checking the power mode.

Example 1

The following example modifies the current fault population in the tool with the contents of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

The example reads in the detected faults from the fault file and applies protection to them, restricting the tool to generating patterns for only undetected faults. The protection saves the time the tool would otherwise spend duplicating the earlier effort, prevents the tool from reclassifying as AU faults the DS faults from the previous session, and results in a smaller pattern set. The tool will include the protected detected faults when it calculates coverage statistics for the ATPG run. Notice the switches used with the report_faults command to display the protected and unprotected faults.

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -protect
report_faults -all
report_faults -protected
report_faults -unprotected
create_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage:

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault  -merge
reset_au_faults -reclassify
create_patterns
report_statistics
write_faults modes_1_and_2.fault -replace
```

Example 3

The following example illustrates the reuse of a faults file in a case that needs the -instance switch.

Tip: As a reminder, the read_faults command automatically adds the faults as well, assuming that they reference faultable sites. The read_faults command should never be followed by the -All command; that sequence of commands is equivalent to only issuing -All and is never what you intended. If you do not execute the add_faults command and only issue the read_faults command, will only target the faults loaded, which is the desired result.

run 1: The user has an initial design that will eventually be part of a larger design. This initial design has a special subset of all stuck-at faults that are created via nofaulting and/or deleting faults. This fault set needs to be reused when this initial design is reused in the larger design. In this run, we assume that the user works and obtains adequate fault coverage. In order to be able to reuse the faults, the faults are written out to a file *stuck_file*.

run 2: The user has embedded the initial design from run 1into a larger design; the instance of the initial design is named *alu1*. The user now wants to reuse the *stuck_file* fault file (created in run 1) to define the faults within that particular instance (*alu1*) of the new larger design.

Assume that a PI of the alu is named *inA*. Note that this will no longer be a faultable site once embedded (no longer a top level IO pin, or on a model boundary).

```
read_fault_sites stuck_file  -instance alu1 -verbose
// Loading faults  for instance alu1.
// Note: Original fault /inA is no longer a faultable site.
// Could not load alu1/inA
// Note: 1 fault not loaded because no longer a faultable site.
```

The first note in the output is omitted if -Verbose is not issued. The summary line in the second note is always issued to indicate how many were not loaded and added due to no longer being a faultable site, unless none were rejected for that reason.

Assume that a fault in the file *dff_32/Q* was loaded and therefore added, but the user wants to delete that fault. Because it is inside *alu1* once loaded, that must also be used in the reference to unambiguously identify the fault.

```
delete_faults alu1/dff_32/Q -s 0
```

Example 4

The following example modifies the current fault population in the tool with the contents of an external fault file, *my_detected_faultlist*. Assume this file is the detected fault list from an earlier session in which patterns were generated and saved to disk along with the detected faults.

The example reads in the detected faults from the fault file and applies protection to them, restricting the tool to generating patterns for only undetected faults. In addition, the -power_check switch loads the faults on a power-on domain and discards the faults in a power-off domain.

set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -protect -power_check on

Example 5

The following example add_faults only for instance ELT_edt_aux_I1from the .flt file, and treats that instance as a graybox:

read_faults det1_all.flt -instance ELT_edt_aux_I1 -graybox -verbose

Example 6

The following example updates the faults in the current fault list with the contents of the external UDFM file *c090_std_io.udfm* and displays parsing errors and warnings:

read_faults /home/design/c090_io/c090_std.io.udfm -verbose

```
//  Warning: UDFM - File c090_io.udfm, line 31 - Cell definition u_and2
does not exist in the circuit.
```

Related Commands

add_faults
delete_faults
read_cpf
read_upf
report_faults
report_testability_data
set_attribute_value
set_fault_mode
set_fault_protection
set_fault_sampling
set_fault_type

read_flat_model

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

read_flat_model ***design_path***

Description

Loads a flattened model into the tool.

The tool can read only one flat model at a time, so you must first use the command before using the read_flat_model command if you previously used this command.

Arguments

- ***design_path***

  A required string that specifies the pathname of a flattened model.

Example

The following example loads a flat model:

```
read_flat_model /u/my_account/designs/model1
```

Related Commands

delete_design
read_verilog
set_current_design
set_design_macros
delete_cell_library
write_flat_model

read_icl

Context: all contexts

Mode: setup

Usage

read_icl **file_name_list** [-bsdl_file *bsdl_file*] [-force]

Description

Reads ICL files into the internal ICL database.

Arguments

- **file_name_list**

  A required list of strings that specifies the names of the ICL files to load. The names of the files can contain wildcards. You can define the list of files to read into a Tcl list and use it to load the files.

  If you have used the command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- -bsdl_file *bsdl_file*

  An optional option value pair that defines the name of a BSDL file. The option is only needed and used when reading an ICL module containing an AccessLink of STD_1149_1_2001 element. The BSDL file must define an "Entity *entity_name* is" clause where *entity_name* is the one specified in the BSDLEntity property of the AccessLink wrapper. Refer to the "Example" on page 332 for an example of the -bsdl_file usage.

- -force

  An optional switch that suppresses the normal error message that is generated when the read_icl command is issued after the ICL elaboration has happened. With the -force option, the ICL elaboration tree is discarded instead and must be recreated afterward using the set_current_design command.

Return Value

None

Example

In the following example, the first read_icl command reads a set of ICL files specified as multiple strings separated by spaces and surrounded by braces. The second read_icl command reads all files matched by the glob command. The third read_icl command reads all filenames contained within the Tcl list icl_list. In all of these cases, undefined ICL modules found within the modules inside the read ICL files are searched for in the specified icl_src directory.

```
set_design_source -format icl -y icl_src -extension icl
read_icl {chip.icl tdr.icl tap.icl block1.icl block2.icl}
read_icl [glob icl/*.icl]
set icl_list [list chip.icl tdr.icl tap.icl block1.icl block2.icl]
read_icl $icl_list
```

Related Commands

delete_icl_modules

read_modelfile

Context: all contexts

Mode: setup

Usage

read_modelfile {***modelfile_name*** | **-Delete**} ***RAM/ROM_instance_name***

Description

Initializes the specified RAM or ROM gate using the memory states contained in the named modelfile or deletes an earlier modelfile assignment.

The read_modelfile command sets the initial memory states of a RAM or ROM gate using the data that you provide in a modelfile. You can create a modelfile from within the library cell or by using the write_modelfile command. The modelfile must contain initialization data that is in the Mentor Graphics modelfile format.

You specify the RAM or ROM gate that you want to initialize by using its instance name. An error condition occurs if the instance contains multiple RAM/ROM gates. When you issue the command, the flattened model may not be available, so the tool checks for the correctness of the instance name and the modelfile name during rules checking.

You may also initialize memory states of a RAM or ROM gate by specifying the modelfile from within the RAM or ROM model description. To do so, use the **init_file** attribute. When using the init_file attribute, you must specify the full pathname to the initialization file if the file is not located in the directory from where you invoked the tool. For more information about modeling RAMs and ROMs and the init_file attribute, refer to the "RAM and ROM" subsection of the *Tessent Cell Library Manual*.

Modelfile Format

A Mentor Graphics modelfile contains addresses and data. You must present the addresses in hexadecimal or binary format. You can specify a range of addresses such as 0-1f. An address range may contain an asterisk (*) wildcard character. For example, to specify that you want all addresses set to a hexadecimal F, use "* / f;". You cannot use an X in an address.

To add comments to a RAM or ROM initialization file, precede the comment text with two forward slashes (//). For example:

```
0/3;
// here is a comment
1/2;
```

You can present the data in either binary or hexadecimal format; the default is hexadecimal. To specify data in binary format, you must add a '%' to the beginning of the data values. If you use an X within hexadecimal data, all four bits that it represents are Xs. Therefore, to set a single bit to X, use the binary format.

The following two examples are equivalent. The first example shows both an address and its associated data in hexadecimal. The second example shows the same address and data, but the data is now shown in binary.

```
ABCD / 123X;

ABCD /%000100100011XXXX;
```

The following is an example of what an initialization file may look like (range 0-1f):

```
0/ a;
1-f / 5;
10 / 1a;
11-1f / a;
```

Assuming that the ROM has 5 data bits per word, the least that can fit the largest data of "*1a*", following this example, the contents of the memory would be:

```
Word (MS-LS)/  Data (MS-LS)
00000       /  01010       // Created by "0/a;"
00001       /  00101       // Created by "1-f/5;"
00010       /  0101
00011       /  00101       ...
...
01111       /  00101       // Created by "1-f/5;"
10000       /  11010       // Created by "10/1a;"
10001       /  01010       // Created by "11-1f/a;"
10010       /  01010       ...
...
11111       /  01010       // Created by "11-1f/a;"
```

You can use an asterisk (*) for an address range. For example, you could rewrite the previous initialization file as shown here. In this example, the first line assigns the data value "*a*" to the full address range 0-1f. The subsequent lines overwrite the "*a*" data value with the new data values for the specified addresses.

```
* / a;
1-f / 5;
10 / 1a;
```

Making the same assumption as the previous example, following this example, the contents of the memory would be:

```
Word (MS-LS)/  Data (MS-LS)
00000       /  01010    // Created by "*/a;"
00001       /  00101    // Created by "1-f/5”; (replaces 'a' from */a)
00010       /  00101
00011       /  00101    ...
...
01111       /  00101    // Created by "1-f/5”; (replaces 'a' rom */a)"
10000       /  11010    // Created by "10/1a”; (replaces ‘a' from */a)"
10001       /  01010    // Created by "*/a;"
10010       /  01010    ...
...
11111       /  01010    // Caused by "*/a;"
```

Arguments

- ***modelfile_name***

  A string that specifies the pathname to and filename of the modelfile containing the RAM or ROM initialization data in Mentor Graphics modelfile format.

- **-Delete**

  A switch that deletes the modelfile initialization assignment you made previously for the specified RAM or ROM gate instance. Use this switch to undo an incorrectly specified modelfile or instance name (due, for example, to a misspelling or a non-existent modelfile or instance).

Note

If you misspelled the RAM/ROM_instance_name when you first assigned the modelfile, you need to use the same spelling when you delete the assignment. After you remove the incorrect modelfile initialization assignment using the -Delete switch, you can re-issue the read_modelfile command using the corrected modelfile or instance name.

- ***RAM/ROM_instance_name***

  A required string that specifies the instance name of the RAM or ROM gate that you want to initialize, or for which you wish to delete a previous modelfile initialization assignment.

Example 1

The following example initializes the memory states of a RAM gate, then performs an ATPG run:

```
read_modelfile model.ram /p2.ram
set_system_mode analysis
create_patterns
```

If the memory primitive specified in the read_modelfile command did not exist, then the set_system_mode analysis command would result in an error message similar to the following:

```
//  Error : /p2.ram given in an earlier read_modelfile command is NOT a
RAM or ROM primitive.
```

Example 2

The next example fixes the error by using the -Delete switch to remove the incorrect entry from the tool's internal list of RAMs and ROMs to be initialized. The example then re-issues the original command using the correct instance name (assumed to be `/p1.ram`):

```
read_modelfile -delete /p2.ram
read_modelfile model.ram /p1.ram
```

Related Commands

create_initialization_patterns (FS)
set_ram_initialization (FS)
write_modelfile

read_patterns

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

Mode: analysis

Usage

read_patterns ***filename*** [-append]
[-mask_file *mask_filename* [-maskfile_pattern_offset *number*]]
[-preserve_masked_observe_points]

For patterns -scan_retargeting

read_patterns ***file_name*** [-append]

Description

Loads the external pattern set from the specified file.

For pattern retargeting, this command extracts the core name and mode from the pattern file and automatically associates the patterns with the correct core and mode.

This command appends to or replaces the external pattern set, and optionally provides additional observation masking information.

Arguments

- ***filename***

  A required string that specifies a file containing the patterns to be read. This file is typically created with the command. The patterns can be in ASCII, binary, STIL, or WGL test pattern format.

- -append

  An optional switch that appends the new patterns on the existing external pattern set. By default, the command deletes the existing external pattern set and replaces with the new pattern set.

- -mask_file *mask_filename*

  An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns, use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

  The tool obtains the masking information from a text file you create, or from the external patterns read in. In the case of external patterns, the tool applies masks wherever Xs appear in a pattern's unload and measure_po values.

This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using “write_patterns -external”, you will not see the masking information from the mask file reflected in the saved patterns; the saved patterns will be exactly the same as the original patterns.

To save a pattern set that incorporates the masking information, use the command to perform a good machine simulation of the external pattern set you specified with “read_patterns -mask_file”. Then use the command *without* the -External switch to write the resulting internal pattern set to a file.

The Mask File

The *mask_filename* specifies a mask file, a text file you create that lists the patterns and locations you want masked. Use the statements and syntax described in to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

The mask file must be in a pattern based format. Most ATE’s return accumulated cycle-based failure data. Use the command to convert cycle-based failure data to pattern based data for the mask file.

Table 7-20. Mask File Statements

Statement	Usage Rules
<location>	Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows: • If “type mask” or “type observe” precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <channel_name> <channel_output_pin_name>

Table 7-20. Mask File Statements (cont.)

Statement	Usage Rules
type <value>	Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values: • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for *all* patterns. Overrides other type statements that cover the same location. Use as many type statements as you like. By default, the tools assume “type mask” for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
//   pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
//   “type observe” for cell 3 of chain 5 for pattern 40, and
//   subsequent “type observe” for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
//    pattern 65.
65 chain5 99
```

- -maskfile_pattern_offset *number*

  An optional switch and integer that specifies the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked. For more information, refer to .

- -preserve_masked_observe_points

  An optional switch that allows you to preserve existing masked observation points (scan cell or primary output) in external patterns loaded into the tool. Even if known values are determined after simulation, the tool continues to mask those locations. You can optionally use -mask_file to specify additional observation points to be masked.

Note

When you use the -preserve_masked_observe_points switch, the tool cannot use multiprocessing.

Example 1

The following example demonstrates how to perform fault grading for stuck-at faults using the transition pattern set created previously. Next, the example shows how to perform top-up ATPG.

set _context patterns -scan
set_system_mode analysis
set_fault_type stuck
add_faults -all
read_patterns pattern_transition
simulate_patterns -store_pattern none
create_patterns
write_patterns pattern_topup.stil -stil

Example 2

The following example demonstrates the use of the -maskfile_pattern_offset switch:

Consider the following mask file, named *mask_data.file*:

```
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
```

The following command reads the patterns from a STIL file, reads the above mask file, applying an offset of 400 in the mask file:

read_patterns pats_400_to_599.stil -mask_file mask_data.file -maskfile_pattern_offset 400
simulate_patterns -store_patterns all
write_patterns pats_400_to_599_masked.stil -stil

The result is that patterns 420 and 425 from the STIL file are masked.

Example 3

The "Retargeting Example" in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Commands

delete_patterns
simulate_patterns
write_patterns

read_procfile

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

Mode: analysis

Usage

read_procfile ***proc_filename***
[-Append_or_timing_update | -Replace]

Description

Reads a test procedure file and merges the timing and named capture procedure data it contains with existing data loaded from previous test procedure files.

This command:

- Replaces each existing timeplate with the definition in *proc_filename* for that timeplate.
- Adds any new timeplates defined in *proc_filename*.
- Updates the timing (timeplates) used by existing scan procedures (like test_setup, load_unload and shift) to match the definitions of these procedures in *proc_filename*.
- Merges the named capture procedure (NCP) data in *proc_filename* with NCP data that exists in the tool's database as summarized in .

Table 7-21. Named Capture Procedure Merging Under Different Conditions

Condition	-Append_or_timing_update	-Replace
New NCPs[1] (no NCP in tool has the same name)	Adds new NCPs	Deletes existing NCPs and adds NCPs from *proc_filename*
NCPs of same name exist in tool but are *not* used[2]	Updates existing NCP's timing only	Deletes existing NCPs and adds NCPs from *proc_filename*
NCPs of same name exist in tool and are used[2]	Updates existing NCP's timing only	Invalid; results in an error message.
No NCP in *proc_filename*	Does not alter existing NCPs	Deletes existing NCPs if not used

1. NCP = named capture procedure
2. Used means at least one pattern in the internal or external pattern database uses the NCP.

- Replaces existing clock control definitions of the same name if no test patterns exist. If test patterns exist, an error displays.
- Adds new clock control definitions.

Tip: A good practice is to regularly use the command to write existing procedures and timing data to a test procedure file when you are experimenting with procedures and timing. This allows you to preserve the most up-to-date procedures and timing data in a form you can use in future runs simply by updating the command in your dofile. It also enables you to recover quickly if existing procedures and timing are altered in an undesirable way.

Arguments

- ***proc_filename***

  A required string that specifies the pathname or filename of the test procedure file to read.

- -Append_or_timing_update

  An optional switch that does the following using the data in *proc_filename*:

  - Adds any new NCPs in *proc_filename* to the tool's existing database of NCPs.

    The tool treats as new any NCP in *proc_filename* whose name is not identical to that of an NCP in the tool's existing database. An added NCP is available immediately for ATPG use unless you disable it with the Off command.

  - Update the timing (timeplate) of existing NCPs and other procedures as applicable, without altering them in any other way.

  This is the default.

Note

An NCP in *proc_filename* and its counterpart of the same name in the tool's internal database must have the same event sequence. If their event sequences differ, the command is aborted.

- -Replace

  An optional switch that deletes all existing NCPs in the tool's internal database and replaces them with the NCPs if any in *proc_filename*, provided a current internal or external pattern set does not contain a pattern that uses an existing NCP. If even one existing NCP is used by a pattern, none of them are deleted or replaced. If *proc_filename* does not contain any NCPs, the existing NCPs are deleted and not replaced, provided a current pattern does not use any of them.

Example 1

Test procedure file *orig.proc* defines a simple timeplate as well as a shift, load_unload, and named capture procedure *cap1*. The following example reads in the procedure file, then lists the

current timeplates and procedures stored in the tool's database (for brevity, only the parts of the timeplates and procedures relevant to this example are shown):

add_scan_groups grp1 orig.proc
set_system_mode analysis
report_timeplate

```
timeplate gen_tp1 =
    ...
end;
```

report_procedures

```
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
 end;

 procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
 end;

 procedure capture cap1 =
    timeplate gen_tp1 ;
    ...
 end;
```

Example 2

Assume a second test procedure file *update.proc* defines a new timeplate in addition to the original one, revises the shift procedure to use the new timeplate, and defines a new named capture procedure *cap2*. The following example adds the new timeplate, updates the timing in the scan procedures, and adds the new named capture procedure. (The timeplate and procedure changes resulting from the read_procfile command are highlighted in bold in the output of the reporting commands.)

read_procfile update.proc

```
//  The following occurred at line 46 in file update.proc:
//  Procedure shift updates timing in same procedure from file orig.proc.
//     (W14)
//  Loaded new procedure file successfully.
//  1 new capture procedure is added.
//  2 total capture procedures are in the system.
//  2 capture procedures are active for test generation.
```

report_timeplate

```
timeplate gen_tp1 =
    ...
end;

// Timeplate gen_tp1 has an old timeplate
// timeplate gen_tp1 =
```

```
//    ...
// end;

 timeplate gen_tp2 =
    ...
 end;
```

report_procedures

```
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp2 ;
    ...
 end;

 procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
 end;

 procedure capture cap1 =
    timeplate gen_tp1 ;
    ...
 end;

 procedure capture cap2 =
    timeplate gen_tp2 ;
    ...
 end;
```

If the read_procfile command of the preceding example had included the -Replace switch, the original NCP would have been replaced with the new one:

read_procfile update.proc -replace

```
//  The following occurred at line 46 in file update.proc:
//  Procedure shift updates timing in same procedure from file orig.proc.
      (W14)
//  Loaded new procedure file successfully.
//  1 new capture procedure is added.
//  1 total capture procedure is in the system.
//  1 capture procedure is active for test generation.
```

report_procedures

```
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp2 ;
    ...
 end;

 procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    ...
 end;
```

```
procedure capture cap2 =
   timeplate gen_tp2 ;
   ...
end;
```

Related Commands

add_scan_groups
report_procedures
report_timeplate
write_patterns
write_procfile

read_sdc

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

Mode: setup, analysis (dft -edt, patterns -scan, patterns -scan_diagnosis and dft -test_points contexts only)

Usage

read_sdc ***sdc_filename***… [{-Instance *ins_pathname*…} | -Module *module_name*] [{> | >>} *file_pathname*]

Description

Reads false path and multicycle path information from a Synopsys Design Constraint (SDC) file.

Typically you determine false and multicycle paths as part of static timing analysis before running ATPG.

The read_sdc command parses the SDC file, looking for the set_false_path command, the set_multicycle_path command, and other appropriate commands and their arguments. A summary message displays the following information for the relevant commands found in the SDC file:

- Number of false paths — Paths that cannot be activated in the design's functional (at-speed) mode of operation or are untestable due to test constraints you have applied. False paths are derived from the SDC command set_false path, which specifies false path timing (the paths that the STA tool should not evaluate for timing).
- Number of multicycle paths — Paths that have a combined time delay through the path greater than the clock period. False paths are derived from the SDC command set_multicycle_path.
- Number of erroneous paths — False paths that are erroneous (for example, due to misnamed pins).
- Number of case analysis — Conditional false paths, which are false paths along with a Boolean condition. If the associated condition is violated, the effect of the false path is considered. Conditional false paths are derived from the Boolean constraints defined by the SDC command set_case_analysis.
- Number of disable timings — Derived from the SDC command set_disable_timing, which disables the specified timing arcs in a circuit and is used to speed up the runtime of the tool. The set_disable_timing command is virtually equivalent to the set_false_path command.

The SDC commands set_false_path, set_multicycle_path, set_case_analysis, and set_disable_timing are converted into equivalent commands. For example, the following SDC command set_disable_timing:

```
set_disable_timing [get_pins {U1/buf2/Z}]
```

would be translated into the following add_false_paths command:

```
add_false_paths -through U1/buf2/Z
```

The following SDC command set_case_analysis (shown in):

```
set_case_analysis 0 [get_pins {U1/select}]
```

Figure 7-29. Case Analysis Example

would be translated into one conditional false path and one unconditional false path:

```
add_false_paths -through U1 if (U1/select !=0)
```

Note

That is, if U_1/select is not 0, the effect of this false path is considered.

```
add_false_paths -through U1/select
```

When test patterns are simulated during ATPG, the related incorrect capture responses are identified and masked (modified to X) in the resultant test patterns, so they are not the source of simulation mismatches when verified in a timing-based simulator.

The read_sdc command supports the following SDC commands:

- create_clock
- create_generated_clock
- set_case_analysis
- set_clock_groups
- set_disable_timing
- set_false_path
- set_hierarchy_separator
- set_multicycle_path

Be aware that the SDC commands create_clock and create_generated_clock are completely independent of the command add_clocks, which you must still use.

Tip: To avoid problems extracting correct paths from the SDC specifications, write out the SDC file using the PrimeTime static timing analysis tool. Follow the steps outlined in the section "Ensuring the SDC File Contains Valid SDC Information" in the *Tessent Scan and ATPG User's Manual*.

For more information, see "Preventing Pattern Failures Due to Timing Exception Paths," in the *Tessent Scan and ATPG User's Manual*.

Tip: You can use the command to list all of the SDC-defined clocks.

Hybrid TK/LBIST Flow Usage

Reads in the SDC file that describes the false and multicycle paths that should be blocked during LogicBIST.

The scan cells that are the destination of false and multicycle paths will be inferred from the SDC file. Re-circulating muxes with inversion will be inserted on their data input pins to ensure these scan cells do not capture values from timing exception paths. Both false paths and multicycle paths are bound in the same way.

Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***sdc_filename***

  A required, repeatable string that specifies the pathname of one or more SDC files to read.

- -Instance *ins_pathname*

  An optional switch and repeatable string that specifies the pathname(s) of one or more design instances to which the information in *sdc_filename* should be applied. Use this switch when the SDC information in the file was developed at the module (block) level and you now want to read it in and have it properly applied to the specified instance(s) of that module in the netlist.

- -Module *module_name*

  An optional switch and string pair that specifies a design module or ATPG library model, to all instances of which the information should be applied in the specified SDC file(s). Use this switch to specify when the SDC information in the file(s) was developed at the module (block) or ATPG library model level and that it should be applied to every instance of that module or model in the netlist.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file_pathname* with a list of any syntax errors encountered while reading the specified SDC file(s).

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname* a list of any syntax errors encountered while reading the specified SDC file(s).

Example 1

The following example reads the file *new_design_sdc* containing one false path definition and one multicycle path definition in Synopsys Design Constraint (SDC) format:

read _sdc new_design_sdc

```
// Reading SDC file new_design_sdc ...
// Finished reading SDC file new_design_sdc.
// read_sdc summary: 1 false path, 1 multicycle path, 0 erroneous paths
```

Tip: If the SDC file contains more than one description of the same path, it is counted only once. Also, the path counts reported in the read_sdc summary apply to the current command only; they are not cumulative counts of the total number of false and multicycle path definitions stored in the internal data base of the tool. To view cumulative data for the path definitions stored within the tool, use the and commands.

Example 2

Continuing with the preceding example, the following example reads another SDC file *latest_design_sdc* containing one false path and three multicycle path definitions, then reports on the false path definitions now stored within the tool:

read_sdc latest_design_sdc

```
// Reading SDC file latest_design_sdc ...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 1 false path, 3 multicycle paths, 0 erroneous paths,
//    0 disable timings, 0 case analysis.
```

report _false _paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2
```

Example 3

The SDC files of the two preceding examples could have been read with one command as in the following example:

read_sdc new_design_sdc latest_design_sdc

```
// Reading SDC file new_design_sdc...
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 2 false paths, 4 multicycle paths, 0 erroneous paths,
//    0 disable timings, 0 case analyses.
```

Notice the read_sdc summary lists the combined totals for all the files read with the one command.

Example 4

Suppose the SDC file *new_design_sdc* of the preceding example contained two false path definitions instead of one, but one of the definitions was erroneous due to misnamed pins. The following example shows sample messaging when this error is detected in the SDC file:

read _sdc new_design_sdc latest_design_sdc

```
// Reading SDC file new_design_sdc ...
//  Error: Incorrect pin/instance path name(s) in SDC file new_design_sdc,
//     line 5.
//  Error: No gate found for "all_inputs all_outputs"
// Finished reading SDC file new_design_sdc.
// Reading SDC file latest_design_sdc...
// Finished reading SDC file latest_design_sdc.
// read_sdc summary: 3 false paths, 4 multicycle paths, 1 erroneous path,
//    0 disable timings, 0 case analyses.
```

report _false _paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B_input -to_clock all_outputs
// Total reported false paths = 3
```

Notice the tool now has the erroneous path definition stored internally. From here, you might want to check for any additional details about this particular definition:

report_false_paths -debug_error -from B_input -to_clock all_outputs

```
// False Path -from B_input -to_clock all_outputs did not find any
//    gate for "B_input all_outputs"
```

Example 5

The following example deletes the erroneous path definition, enters a corrected version of the definition, then confirms there are no remaining errors in any of the false or multicycle path definitions:

delete _false _paths -from B_input -to_clock all_outputs

```
// Deleted 1 false path
```

report_false_paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// Total reported false paths = 2
```

add_false_paths -from B -to y[2] y[1] y[0]
report_false_paths

```
// False Path -from_clock clk1 -to_clock clk2
// False Path -from clk2 reset scan_in1 -to y[2] y[1] y[0]
// False Path -from B -to y[2] y[1] y[0]
// Total reported false paths = 3
```

report_false_paths -error

```
// Total reported false paths = 0
```

Example 6

The following example reads the SDC file *my_block_sdc* and applies the information it contains to instance U1 of the design module *my_block*:

read_sdc my_block_sdc -instance U1

```
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 2 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses.
```

Example 7

The following example reads the file of the preceding example and applies the SDC information to instances U2 and U3 of the same design module:

read _sdc my_block_sdc -instance U2 -instance U3

```
// Reading SDC file my_block_sdc ...
// Finished reading SDC file my_block_sdc.
// read_sdc summary: 4 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses.
```

Example 8

The following example reads the SDC file *my_block_sdc* and applies the information to every instance of the design module *my_block*:

delete _false _paths -all
read _sdc my_block_sdc -module my_block

```
// Reading SDC file my_block_level_sdc ...
// Finished reading SDC file my_block_level_sdc.
// read_sdc summary: 6 false paths, 0 multicycle paths, 0 erroneous paths
//    0 disable timings, 0 case analyses.
```

Example 9

This example shows that we can read an SDC file in setup mode, transition to analysis mode, and perform X-bounding based on the paths in the SDC file:

read_sdc my_design.sdc

set_system_mode analysis

analyze_xbounding

Related Commands

add_false_paths
report_clocks
report_false_paths
report_multicycle_paths

read_sdf

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

Mode: setup, analysis

Usage

read_sdf ***sdf_filename***...
[-MInimum_delay | -TYpical_delay | -MAximum_delay]
[-Replace | -Append | -INCremental]
[-INstance *hierarchical_instance_name*]

Description

Loads timing data from a Standard Delay Format (IEEE SDF P1497 Draft 0.10) file.

The SDF file is generated by your static timing analysis tool. For complete information, see "Using Timing-Aware ATPG, in the *Tessent Scan and ATPG User's Manual*.

The following limitations apply to the SDF file:

1. For SDF conditional delays, the tool will load them into an internal database. However, only the maximum delay will be used for calculating actual delay. For example:

   (COND b (IOPATH a y (0.21) (0.54)))
   (COND ~b (IOPATH a y (0.27) (0.34)))

   Timing-aware ATPG uses (0.27) (0.54) regardless of the value of b.

2. Only the first two delay value tokens are used.
3. PATHPULSE, PATHPULSEPERCEN, DEVICE, and RETAIN statements are ignored.
4. Delays on Memory and Latch are ignored.
5. Hold time information is ignored.
6. Delays in clock lines are ignored.
7. Latch is assumed to be either a launch or capture point, not as a combinational logic in-between.

Arguments

- -MInimum_delay

  An optional switch that loads only the SDF minimum delay data into the tool.

- -Typical_delay

  An optional switch that loads only the SDF typical delay data into the tool. This is the default.

- -MAximum_delay

  An optional switch that loads only the SDF maximum delay data into the tool.

- -REplace

  An optional switch that removes any previously loaded SDF file information before loading the new SDF file information.

- -APpend

  An optional switch that loads the new SDF file information in addition to the existing information. This is the default. This switch allows you to load multiple SDF files for different modules.

  The tool will issue an error message if the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file. Use the -INCremental switch if you want to merge all delay data for a flattened pin.

Note

When using this switch, if the selected delay type (minimum, typical or maximum delay) is different from the previous read_sdf command, all previously loaded SDF timing information will be discarded.

- -INCremental

  An optional switch that loads the new SDF file information in addition to the existing information. If the new SDF file contains delay data for a flattened pin that was already defined by a previously loaded SDF file, the new delay data will be cumulated with the previous delay data for that pin. Use the -APpend switch if you do not want to merge any delay data for a flattened pin.

- -INstance *hierarchical_instance_name*

  An optional switch and string pair that specifies the hierarchical instance name to which the SDF file is associated. This option is useful when the SDF file is at the module level.

Examples

The following example reads the typical delay data from the *top_worst.sdf* file:

```
read_sdf top_worst.sdf
```

The following example reads the delay data for multiple modules:

```
read_sdf file1.sdf -maximum_delay -instance /top/module1
read_sdf file2.sdf -maximum_delay -append -instance /top/module2
read_sdf file3.sdf -maximum_delay -append -instance /top/module3
```

Related Commands

report_statistics

set_atpg_timing

read_upf

Context: all contexts

Mode: setup

Usage

read_upf ***filename***

Description

Loads a UPF (Universal Power Format) file and removes any previously loaded power data.

You can load multiple UPF files using the native UPF “load_upf” command.

The tools support UPF version 2.0.

For information about what happens when you load power data, refer to .

Arguments

- ***filename***

  A required string that specifies the name of the UPF file.

Example 1

The following example loads a UPF file:

read_upf upf_file

Related Commands

add_faults
delete_faults
report_faults
read_cpf
report_power_data

read_verilog

Context: all contexts

Mode: setup

Usage

read_verilog [***design_path***...] [-verbose] [-force | -no_overwrite] [-blackbox]
[-in_library *logical_lib_name*] [-no_lib_rescan]
[-mfcu] [-format {1995 | 2001 | sv31a | sv2005 | sv2009}]
[-pragma_list *pragma_list*] [-ignore_synthesis_off_sections {On | Off}]
[-vcs_compatibility] [-f *file*]

Description

Reads one or more Verilog files into the specified or default logical library.

The list of available libraries is defined with the , which you only use in rtl context; only one library exists in no_rtl context and it is named “work”.

Note that the MGC_TESSENT and SYNTHESIS design macros are defined by default. For more information, refer to the command description.

Arguments

- ***design_path***

  A required, repeatable string that specifies the name of a Verilog file to load. The names of the files can contain wildcards. You can define the names of files to read into a Tcl list, and use the name of the Tcl list as the design_path. The design_path is optional if you use the -f switch to specify a file that contains at least one design_path.

  If you have used the set_design_sources command, child modules found instantiated into the currently read modules are searched for in the specified -y directories and the -v files.

- -verbose

  An optional switch that reports each file read. For every file read, including the ‘include files, the following messages are returned:

  ```
  // Reading Verilog file filename
  // Finished reading file filename
  ```

- -force

  An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), executing the read_verilog command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you

execute the command with the -force switch to indicate that it is acceptable to delete the setup data.

- -no_overwrite

  An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- -blackbox

  An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters. (Note, however, that the tool parses the internals of the modules to ensure that they contain valid Verilog code.) The resulting module(s) is a blackboxed module. RTL can be read in the no_rtl context when this switch is used. The -blackbox switch is useful mainly to read the interface of an RTL module and be able to instantiate the module in the design using the no_rtl context.

Note

The write_design command does not write out blackboxed modules. If you want to write out the interface view of a module, read the module without the -blackbox option and write it out using the " -interface" option.

- -in_library *logical_lib_name* ("dft -rtl" context only)

  An optional switch that specifies in which logical library the modules are to be read into. The list of possible logical libraries including the default library is defined by the command. If this option is not specified, the modules are read into the default library.

- -no_lib_rescan

  An optional switch that specifies to not rescan the -y directories and the -v files from the beginning each time, but to search for the module beginning with the directory or file in which the previous module was found.

- -mfcu ("dft -rtl" context only)

  An optional switch that applies to SystemVerilog formats which specifies to make global declarations found in one file visible to all SystemVerilog files read in afterward. By default, global declarations found in one file are only visible for elements found later in the same file.

- -format {1995 | 2001 | sv31a | sv2005 | sv2009} ("dft -rtl" context only)

  An optional switch that specifies the format of the input Verilog files. The default is 2001. In a mixed-format situation, you should be sure to re-define your -y and -v library search path when loading modules in a new format because you cannot mixed formats within a single read_verilog command invocation.

  The sv31a is not a standard format but it is the default inside Design Compiler when hdlin_sv_packages is not defined.

- -pragma_list *pragma_list*

  An optional switch and string pair that defines the pragmas for the input files. The pragma_list is a Tcl list containing one or more objects. When this option is not specified, the following list of strings is used as default pragmas: {pragma synthesis mgc_tessent synopsys}. You can overwrite the default pragmas by specifying a new pragma list as shown here:

  read_verilog 1.v -pragma_list {my_pragma1 mgc_tessent}

  You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

  read_verilog 1.v -pragma_list {}

- -ignore_synthesis_off_sections {<u>On</u> | Off}

  An optional switch and literal pair that specifies whether to ignore the synthesis_off sections in the input files. The default is On.

  <u>On</u> — A literal that specifies to ignore synthesis_off sections and to NOT compile the code found within the synthesis_off/on block.

  Off — A literal that specifies to not ignore the synthesis_off sections and to compile the code found within the synthesis_off/on block.

- -vcs_compatibility (“dft -rtl” only)

  An optional switch that specifies to ignore a series of syntax errors allowed by the Synopsys compilers. By default, the tool reports these non-compliant usages as errors.

  When specified, this setting is in effect until the delete_design command is executed. This switch can only be specified in the first execution of the read_verilog command after tool invocation, or after the delete_design command is executed.

- -f *file*

  An optional switch and string pair that specifies an ASCII file that contains pointers to design files. The referenced file is usually generated by another tool, so the -f option allows you to conveniently use the file rather than to reformat the contents of the file into a read_verilog command line. The file can contain only the following elements, one per line:

  - *design_path* — A string that specifies the name of a Verilog file to load.
  - -v *design_path* — A string that specifies the name of a Verilog file to load.
  - -y *dir_path* — A string that specifies a path to a directory that contains Verilog files to load.
  - +libext+*file_extension* — A string that specifies which files in the “-y dir_path” to load. For example: “+libext+.v” loads all .v files.

Example 1

The following example loads all Verilog files matching *.v inside the directory src:

read_verilog src/*.v

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the edited_design directory.

```
SETUP> set_context dft -rtl

# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work will be mapped to a scratch dir
SETUP> set_logical_design_libraries -logical_names { library1 work}

# Define macro1 for Verilog compilation
SETUP> set_design_macros macro1=5

# Search for Verilog modules in /design/abc/*.v
SETUP> set_design_sources -y /design/abc -extension v

# Read Verilog files
SETUP> read_verilog { top.v core.v } -in_library work -format 2001

# Read VHDL files
SETUP> read_vhdl core1.vhdl  -in_library work -format 1993
SETUP> read_vhdl { core2.vhdl core3.vhdl } -in_library library1 -format 1993

# Set the top
SETUP> set_current_design top

# Enter insertion mode
SETUP> set_system_mode insertion

# Perform design query and manipulation commands
INSERTION> ...

# Write the design
INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit
```

Related Commands

delete_design
read_cell_library
read_flat_model
set_current_design
read_vhdl
set_design_macros
set_design_sources
set_design_include_directories
set_logical_design_libraries

read_vhdl

Context: all contexts

Mode: setup

Usage

read_vhdl [***design_path***...] [-blackbox] [-force | -no_overwrite] [-in_library *logical_lib_name*] [-format {1987 | 1993 | 2002 | 2008}] [-pragma_list *pragma_list*] [-ignore_synthesis_off_sections {On | Off}] [-f *file*]

Description

Loads one or more VHDL files.

You can execute this command with the -force switch but without a design_path argument to clear design setup data only and not read any new files.

Arguments

- ***design_path***

  A required, repeatable string that specifies the name of a VHDL file to load. The names of the files can contain wildcards. You can define the names of files to read into a Tcl list, and use the name of the Tcl list as the design_path. The design_path is optional if you use the -f switch to specify a file that contains at least one design_path.

  The command contains an example that shows a suggested way of loading different VHDL files into different libraries.

- **-blackbox**

  An optional switch that instructs the tool to ignore the internals of all of the modules specified in the filename argument and extract only the module port definitions and parameters. The -blackbox switch is mandatory when in non-RTL mode. The resulting module(s) is a blackboxed module skeleton module. RTL can be read when this switch is used. The -blackbox switch is useful mainly to read the interface of an RTL module to instantiate the module in the design. Although the -blackbox switch supports reading parameterized modules, currently you can only insert an instance of such a module with default values.

  Note

  The write_design command does not write out blackboxed modules.

- -force

  An optional switch that suppresses the error message that normally prevents the accidental deletion of previously defined setup information. After you read in a design using this command, you can incrementally issue the command again to read more files. However, once the tool builds the design using set_current_design and you add any configuration information to it (for example, add_clocks or add_scan_chains), executing the read_vhdl

command causes the setup information to be discarded. In this case, to prevent configuration information from being discarded unintentionally, the tool requires that you execute the command with the -force switch to indicate that it is acceptable to delete the setup data.

- -no_overwrite

  An optional switch that instructs the tool not to replace existing modules. If a module is read that has been read before, the tool reports an error. You can use this switch to avoid overwriting modules or accidentally deleting the design.

- -in_library l*ogical_lib_name* (“dft -rtl” only)

  An optional switch that specifies which logical library the modules are to be read into. The list of possible logical libraries, including the default library, is defined by the command. When this option is not specified, the modules are read into the default library.

- -format {1987 | 1993 | 2002 | 2008}

  An optional switch that specifies the expected format of the input files and which VHDL language constructs can be understood during parsing. The default is 1993.

- -pragma_list *pragma_list*

  An optional switch and string pair that defines the pragmas for the input files. The pragma_list is a Tcl list containing one or more objects. When this option is not specified, the following list of strings is used as default pragmas: {pragma synthesis mgc_tessent synopsys}. You can overwrite the default pragmas by specifying a new pragma list as shown here:

  read_vhdl 1.v -pragma_list {my_pragma1 mgc_tessent}

  You can create an empty list of pragmas by specifying an empty Tcl list as shown here:

  read_vhdl 1.v -pragma_list {}

- -ignore_synthesis_off_sections {<u>On</u> | Off}

  An optional switch and literal pair that specifies whether to ignore the synthesis_off sections in the input files. The default is On.

  <u>On</u> — A literal that specifies to ignore synthesis_off sections and to NOT compile the code found within the synthesis_off/on block.

  Off — A literal that specifies to not ignore the synthesis_off sections and to compile the code found within the synthesis_off/on block.

- -f *file*

  An optional switch and string pair that specifies an ASCII file that contains pointers to design files. The referenced file is usually generated by another tool, so the -f option allows you to conveniently use the file rather than to reformat the contents of the file into a read_vhdl command line. The file can contain only the following elements, one per line:

  - *design_path* — A string that specifies the name of a VHDL file to load.

Example 1

The following example instructs the tool to read only the interface of the modules/entities and ignore the contents:

```
set_context dft -no_rtl
read_verilog  data/pll.v -blackbox
read_vhdl  data/ram_bist.vhd  data/edt_ip*.vhd -blackbox
```

Example 2

The following example loads a mixed-language design in the dft context in RTL mode. The results files are written into the edited_design directory.

```
SETUP> set_context dft -rtl

# Specify LIBRARY1 to be mapped into <workdir>/lib1
# work will be mapped to a scratch dir
SETUP> set_logical_design_libraries -logical_names { library1 work}

# Define macro1 for Verilog compilation
SETUP> set_design_macros macro1=5

# Search for Verilog modules in /design/abc/*.v
SETUP> set_design_sources -y /design/abc -extension v

# Read Verilog files
SETUP> read_verilog { top.v core.v } -in_library work -format 2001

# Read VHDL files
SETUP> read_vhdl core1.vhdl  -in_library work -format 1993
SETUP> read_vhdl { core2.vhdl core3.vhdl } -in_library library1 -format 93

# Set the top
SETUP> set_current_design top

# Enter insertion mode
SETUP> set_system_mode insertion

# Perform design query and manipulation commands
INSERTION> ...

# Write the design
INSERTION> write_design -output_directory edited_design -modified

INSERTION> exit
```

Related Commands

delete_design
read_icl
read_verilog
set_current_design
delete_design

read_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Usage

read_visualizer_preferences ***filename***

Description

Reads a DFTVisualizer preferences file and sets current preferences as described in the file.

The read_visualizer_preferences command reads a DFTVisualizer preference file previously saved with the command and updates the current DFTVisualizer session to use the preferences in the specified file.

Tip: When DFTVisualizer first opens in a tool session, it reads the *$HOME/.DftVisualizerrc* preference file if it is present and sets the preferences accordingly; otherwise, it uses hard-coded defaults. Current preference settings are written to this file when you click the Write Prefs button at the bottom of the DFTVisualizer Preferences window.) DFTVisualizer uses the preferences in the default file until overridden by a read_visualizer_preferences command. The last such command entered establishes the preferences DFTVisualizer uses.

Arguments

- ***filename***

  A required string that specifies the name of the preference file to read.

Examples

The following example reads the DFTVisualizer preference file, *my_prefs.dftvis*, and updates the current tool session to use the preferences specified in the file:

```
set_system_mode analysis
read_visualizer_preferences my_prefs.dftvis
```

Related Commands

write_visualizer_preferences

read_window_contents

Context: all contexts

Mode: setup, analysis

Usage

read_window_contents ***file_pathname*** **-display** ***window_name*** [-format *format*]

DFTVisualizer Menu Path:
File > Save As

Description

Reads the contents of the specified file into a DFTVisualizer window.

The file specified by the file_pathname argument must be in a supported format as described in the -format argument description.

The window specified by the window_name argument must be visible and not minimized.

Arguments

- ***file_pathname***

  A required string that specifies the pathname of the file that is to be read into a DFTVisualizer window. If the file is to be read into a Debug or Design window, the file must be in the Mentor Graphics Tessent schematic format.

- **-display** ***window_name***

  A required switch and literal pair that specifies the DFTVisualizer window into which the file_pathname content is to be read. Select the window name from the following list:

  DEBug
  A literal that specifies the Debug window. This is the default.

  DESign
  A literal that specifies the Design window.

  Text_editor
  A literal that specifies the Text Editor window.

- -format *format*

  An optional switch and literal pair that specify the format of the file to be read. The file displays in the Text Editor window with appropriate highlighting and formatting. Format options include:

  TEXt
  A literal that specifies an ASCII text format. This is the default.

  Verilog
  A literal that specifies the Verilog format.

  VHdl
  A literal that specifies the VHDL format.

TESt_procedure
A literal that specifies the Test Procedure format.

Dofile
A literal that specifies the Dofile format.

Examples

Related Commands

close_visualizer
open_visualizer
set_visualizer_preferences
write_window_contents

register_attribute

Context: unspecified, all contexts

Mode: all modes

Usage

register_attribute **-name** ***attribute_name*** **-value_type** ***data_type*** **-default** ***default_value*** [-min *min_value*] [-max *max_value*] [-enum *enum_values]* **-object_types** ***type_list*** [-dense] [-description *description_text*] [-show_default]

Description

Registers a new user-defined attribute.

This command allows you to define new attributes for object types available in Tessent Shell. After registration, the value of the new attribute can be set and retrieved on objects such as a module or an instance.

If an attribute with the same name already exists for the specified object type, or if an object type in type_list does not exist, or if a data type is invalid for one of the options, an error message displays.

If you want an attribute set on a module to be visible (inherited) on the instances of the module, register the attribute only on the *module* object type and not on the instance object type. The same is true for inheritance between ports and pins and between nets and pins. The following exceptions to this inheritance rule exist for Boolean attributes:

- For Boolean attributes registered on module and instance object types — When the value is queried on an instance and it is the default value (usually false), the value from the corresponding module is returned.
- For Boolean attributes registered on port and pin object types — When the value is queried on a pin and it is the default value (usually false), the value from the corresponding port is returned.

Arguments

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute to register. An attribute name must begin with an alpha character and must only contains alpha-numeric characters and the underscore (_).

- **-value_type** ***data_type***

  A required switch and value pair that specifies the attribute's data type. The value_type argument is an enumerated list with the following possible values: Boolean, integer, double, string, enum. For more information on these data types, see “.”

- **-default *default_value***

  A required switch and value pair that specifies the default value for the attribute when the attribute is not explicitly set. This pair is required for attributes with non-Boolean data types only; it cannot be set for attributes whose data type is Boolean. Attributes with a data type of Boolean automatically default to false.

- -min *min_value*

  An optional switch and value pair that specifies the minimum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is greater than or equal to min_value. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see “.”

- -max *max_value*

  An optional switch and value pair that specifies the maximum value for the attribute. When this argument is specified, the value of the attribute is only accepted if it is less than or equal to max_value. This argument is only valid for attributes whose data type is integer or double. For more information on these data types, see “.”

- -enum *enum_values*

  An optional switch and value pair that specifies a Tcl list of possible values for the attribute. The enum_values value can only be set to one value in the list. This argument is only valid for attributes whose data type is string.

- **-object_types *type_list***

  A required switch and value pair that specifies the object type to which the new attribute can be assigned. The type_list string is a Tcl list that contains one or more of the following object types: module, instance, port, pin, or net. If an object type is not included in the type_list, the new attribute cannot be assigned to that object_type.

  Extra data types will be available in subsequent releases and the type_list will be augmented as they become available.

- -dense

  An optional Boolean switch that specifies the storage type for the attribute.

  When not specified — Specifies *sparse* storage which only stores values for objects that have an attribute value set to something different than the default value. This option is a good choice when many of the attributes on an object are set to their default value because it saves memory. However, indexing and searching can be slower and end up consuming more memory if the number of objects having the attribute specified is more than 10% of the object count.

  When specified — Specifies *dense* storage which creates an entry for each object and its attribute value. This option is a good choice when many objects with the same attribute have a distributed set of attribute values because it enables faster indexing and searching. However, it has a higher memory usage when less than 10% of the objects have the attribute specified.

- -description description_text

  An optional switch and value pair that specifies to store a string describing the attribute. The description is displayed when the report_attributes command is executed for the attribute. The description_text argument is a quoted string that can be formatted as a Tcl string which includes support for escaped characters such as “\n” and “\t”.

- -show_default

  An optional Boolean switch that specifies how the registered attribute is handled by the get_attribute_list and report_attributes commands.

  false — If set to false during registration, the get_attribute_list obj_spec and report_attribute obj_spec command executions list or report attributes found on any object within obj_spec *only if* the attribute is set to a non default value.

  true — If set to true, the attribute is always listed or reported if obj_spec contains at least one object of the type for which the attribute is registered for.

Example 1

The following example registers the attribute named speed with a range of possible values equal to or greater than 100 and less than or equal to 500; assigns a default value of 200 to it; sets it as an attribute for the pin design object type only. Notice how the Tcl append command is used to allow typing the string on multiple lines without inserting extra white spaces into the string.

```
set     desc “This attribute is used to hold the target frequency the pin is \n”
appenddesc “allowed to have.\n”
appenddesc “The allowed frequency range of 100 to 500 was chosen to match\n”
appenddesc “the technology spec.”

register_attribute -name speed \
        -value_type integer \
        -default 200 \
        -min 100 \
        -max 500 \
        -object_types pin \
        -description$desc
```

Example 2

The following example defines the attribute named added_by as a string with a value of either “John”, “Patrick”, or “Susan” with a default of John, and a design object type of module:

```
register_attribute -name added_by \
        -value_type string \
        -default John \
        -enum {John Patrick Susan} \
        -object_types net
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
reset_attribute_value
report_attributes
set_attribute_options
set_attribute_value
unregister_attribute

register_tcl_command

Context: unspecified, all contexts

Mode: all modes

Usage

register_tcl_command ***command_name***
-syntax_bnf ***syntax_bnf***
-tcl_proc *tcl_proc*
-context_list *context_list*
-system_mode_list *system_mode_list*
[-resource_list *resource_list*]
[-extra_help *extra_help*]
[-integer_list_option *integer_list_option*]
[-string_list_option *string_list_option*]
[-real_list_option *real_list_option*]
[-integer_list_value *integer_list_value*]
[-string_list_value *string_list_value*]
[-real_list_value *real_list_value*]
[-boolean_option *boolean_option*]
[-integer_option *integer_option*]
[-string_option *string_option*]
[-real_option *real_option*]
[-integer_value i*nteger_value*]
[-string_value *string_value*]
[-real_value *real_value*]

Description

Registers a Tcl command and makes it available in the global name space.

The registered command appears as a normal tool command with proper option parsing and online help messages. The body of the registered command is implemented in a Tcl proc referenced by the -tcl_proc option. After the options are validated, they are packaged into a Tcl list of option-value pairs that are packed into a Tcl array for easy random access as shown in .

You control the availability of the newly registered command using the -context_list and -system_mode_list options.

Arguments

- ***command_name***

  A required string that specifies the name of the command to register. Because the command will exist in the global name space, its name must be unique so that it does not collide with other already registered commands. You can unregister any registered Tcl command using until you execute the lock_current_registration command. After you execute , all already registered commands and attributes are locked and can no longer be modified.

All commands loaded with an automated plugin mechanism are locked before the interactive shell is released to the user. Example 2 explains how to initially debug your commands interactively when it is still possible to unregister them, and then shows you how to package them into a plugin directory which is automatically loaded at tool invocation. For more information on plugins, see the command.

- -syntax_bnf ***syntax_bnf***

  A required switch and string value pair that specifies the grammar rules of your command using simple Backus-Naur Form (BNF). You should place the syntax_bnf string in braces to avoid needing to escape square brackets which are used to express that an option or value is optional. You should use parentheses to group options that are exclusive to others. You can use the | symbol to denote exclusivity of options. BNF uses the symbols you defined when declaring the *option and *value arguments; the symbols must always be enclosed in <> characters.

  The following example defines five arguments; the first two are required and the last three are optional. The fifth argument is mutually exclusive of the third and fourth arguments. An argument is repeatable if its tag is declared with an * as its last character.

  { <arg1> <arg2> ([arg3] [arg4]) | [arg5] }

- **-tcl_proc *tcl_proc***

  A required switch and string pair that specifies the name, including its namespace, of the Tcl proc this command will call. To facilitate packaging the new command as a plugin that is automatically loaded into local namespaces, you should specify tcl_proc using the following format: [namespace current]::*proc_name*. This format ensures that the registered Tcl command always knows where to look for the Tcl proc regardless of which namespace the command is evaluated in.

  For easy interactive debugging that can be easily deployed as a plugin, you should package both your Tcl proc and the register_tcl_command into a single file and surround it with a namespace eval *name_space* {} wrapper. Do not put :: in front of the specified name_space string so that this namespace string will be automatically appended to the one automatically created for all plugins. See the examples for more information on how to do this.

- **-context_list *context_list***

  A required switch and string pair that specifies a Tcl list of contexts the command is enabled in. Each element of the Tcl list is a context with an optional sub-context option as shown here:

  ```
  -context_list {{dft} {patterns -ijtag}}
  ```

  The available contexts are: dft, patterns. The available sub-contexts are those described for each context on the command reference page.

- **-system_mode_list *system_mode_list***

  A required switch and string pair that specifies a Tcl list of system modes the command is to be enabled in. The available modes are: setup, analysis, and insertion.

- -resource_list *resource_list*

  An optional switch and string pair that specifies the list of resources required for the command to be available; if any of the specified resources is not available, the command is not available. The resource_list argument is a Tcl list of strings that identify the resources required. Possible values are:

  design_is_elaborated

  Used to check that the hierarchical data model has been created using the set_current_design command.

  design_modules_exist

  Used to check that at least one Verilog or VHDL module exists in memory and can be accessed using the get_modules command.

  flat_model_exists

  Used to check that the flat data model is available. The flat data model was created by either the create_flat_model command or by the “set_system_mode analysis” command.

  icl_is_elaborated

  Used to check that the ICL hierarchical data model exists. It exists after the set_current_design command if an ICL module corresponding to the current design existed prior to issuing the command. It also exists in the analysis mode if the top-level ICL module was successfully extracted with ICL extract.

  icl_modules_exist

  Used to check that at least one ICL module exists in memory and can be accessed using the get_icl_modules command.

  rtl_mode_enabled

  Used to check that the -rtl option switch was used when specifying the context and that read-in Verilog and VHDL files can contain RTL constructs.

- -extra_help *extra_help*

  An optional switch and string pair that enables you to define a multi-line help paragraph that is displayed at the bottom of the online help. You can use \n to break your strings into multiple lines. The easiest way to create a multi-line help string is to construct it line by line as shown here:

  ```
  set extra_help “This is line1\n”
  append extra_help “and this is line 2\n”
  append extra_help “and this the last line.”
  register_tcl_command -extra_help “$extra_help” ...
  ```

- -integer_list_option *integer_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of integers. The integer_list_option_list string is a Tcl list with three required elements and five optional elements as shown here:

```
{{<symbol>} {-option_name} {tag} [{description_string}] [{range}]
[{enum_list}] [{<help_symbol>}][{<visibility>}]}
```

<symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also displays in the online help description. The symbol string must always begin with a < character and end with a > character.

-option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (i.e. tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

```
{{range,0}{value1}{range,1}{value2}{range,2}{value3}}
```

description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest allowed value is a MIN_INT. When the second integer is not specified, the largest allowed value is MAX_INT. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

enum_list — The sixth element is optional. It allows you to specify strings as allowed values. For example, you may want to define an option that allows a positive integer or the literal “last”. This is achieved by using a minimum value of 0 and an enum string called “last”.

<help_symbol> — The seventh element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

visibility — The eighth element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the

option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- -string_list_option *string_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of strings. The string_list_option is a Tcl list with three mandatory elements and four optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}] [{enum_list}]
  [{<help_symbol>}] [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also displays in the online help description. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

  tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (i.e. tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

  ```
  {{name,0}{value1}{name,1}{value2}{name,2}{value3}}
  ```

  description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

  enum_list — The fifth element is optional. It allows you to specify strings as allowed values. When omitted or specified as an null string, the string_list_option allows any string values.

  <help_symbol> — The sixth element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf string. For example, the help message of the get_instances command shows “[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

  visibility — The seventh element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the

option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- -real_list_option *real_list_option*

  An optional switch and string pair that defines an argument that is an option-value pair where the allowed value is a Tcl list of real numbers. The real_list_option is a Tcl list with three required elements and five optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}] [{range}]
  [{enum_list}] [{<help_symbol>}] [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required. It defines a unique symbol for the argument that you use in the -syntax_bnf syntax_bnf string to refer to it and it also displays in the online help description. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required. It defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name <symbol>”.

  tag — The third element of the list is required. It defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc. You can use an * character as the last character of the tag string to denote that it is repeatable. If the argument is not repeatable (i.e. tag does not end with an *), the Tcl list passed to the Tcl proc will include the tag string followed by the value specified for it. If the argument is repeatable, the tag will be appended with “,#” where # is an integer incremented from 0 to uniquely identify each instance of the argument. For example, if the tag is specified as range* and the argument is specified three times, the Tcl list passed to the Tcl proc will include those six elements as shown here:

  ```
  {{range,0}{value1}{range,1}{value2}{range,2}{value3}}
  ```

  description_string — The forth element is optional. It can be used to define a one-line description for the argument that displays in the online help. If you need to specify the range or enum_list element but do not want to specify a description string, specify this fourth element as empty by using an empty brace pair.

  range — The fifth element is optional. It consists of one or two integers specifying the minimum and maximum value for the argument. When the range is not specified or is specified as empty, the smallest allowed value is a MIN_DOUBLE. When the second interger is not specified, the largest allowed value is MAX_DOUBLE. If you need to specify the enum_list element but do not want to limit the range, specify this fifth element as empty using an empty brace pair.

  enum_list — The sixth element is optional. It allows you to specify strings as allowed values. For example, you may want to define an option that allows a positive integer or the literal “last”. This is achieved by using a minimum value of 0 and an enum string called “last”.

  <help_symbol> — The seventh element is optional. It allows you to specify a different symbol for the option to display in the help message. By default the <symbol> specified as the first element of the list is used in the help message. However, the <symbol> must be unique across all options as it is referenced in the syntax_bnf

string. For example, the help message of the get_instances command shows “[-parent_of_instances <instance_objects> | -below_instances <instance_objects>]”. The symbol for each option has to be different such as <inst_for_parent_of> and <inst_for_below> but the display symbol in the help message can be made to be <instance_objects> for both by specifying the eight element.

visibility — The eighth element is optional. It allows you to specify if you want the option to be hidden from the help message. You use a null string if you want the option to show up in the help message and you use the string “hidden” if you want the option to be hidden.

- -boolean_option *boolean_option*

  An optional switch and string pair that defines an argument that is an option without a value. The boolean_option string is a Tcl list with three required elements and two optional elements as shown here:

  ```
  {{<symbol>} {-option_name} {tag} [{description_string}] [{<visibility>}]}
  ```

  <symbol> — The first element of the list is required and defines a unique symbol for the argument. You use this element in the -syntax_bnf syntax_bnf string to refer to it. The symbol string must always begin with a < character and end with a > character.

  -option_name — The second element of the list is required and defines a unique option name. The option name must start with a - character. The online help displays this option as shown here: “-option_name”.

  tag — The third element of the list is required and defines a unique tag that is used to identify the argument value when invoking the associated Tcl proc.

  description_string — The forth element is optional and can be used to define a one-line description for the argument that displays in the online help.

  visibility — The fifth element is optional and can be used to specify whether the option is included in the help message. You specify a null string "" if you want the option to show up in the help message and you specify the string "hidden" if you want the option to be hidden.

- -integer_list_value *integer_list_value*

  An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

  The integer_list_value string is a Tcl list with two required elements and five optional elements. This string has exactly the same definition as the integer_list_option string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with an * character because only options can be defined as repeatable.

  ```
  {{<symbol>} {tag} [{description_string}] [{range}] [{enum_list}]
  [{<help_symbol>}] [{<visibility>}]}
  ```

- -string_list_value *string_list_value*

  An optional switch and string pair that is identical to the -string_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

The string_list_value string is a Tcl list with two required elements and four optional elements. This string has exactly the same meaning as the string_list_option string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with an * character as only options can be defined as repeatable.

```
{{<symbol>} {tag} [{description_string}] [{enum_list}] [{<help_symbol>}]
[{<visibility>}]}
```

- -real_list_value *real_list_value*

  An optional switch and string pair that is identical to the -real_list_option switch and string pair except that it defines a value as opposed to an option-value pair.

  The real_list_value string is a Tcl list with two required elements and five optional elements. This string has exactly the same meaning as the real_list_option string except that the second element does not exist because there is no option name to define. The tag is not allowed to end with a * as only options can be defined as repeatable.

```
{{<symbol>} {tag} [{description_string}] [{range}] [{enum_list}]
[{<help_symbol>}] [{<visibility>}]}
```

- -integer_option *integer_option*

  An optional switch and string pair that is identical to the -integer_list_option switch and string pair except that only a single integer is allowed as the value as opposed to a Tcl list of integers.

- -string_option *integer_option*

  An optional switch and string pair that is identical to the -string_list_option switch and string pair except that only a single string is allowed as the value as opposed to a Tcl list of strings.

- -real_option *real_option*

  An optional switch and string pair that is identical to the -real_list_option switch and string pair except that only a single real is allowed as the value as opposed to a Tcl list of real numbers.

- -integer_value *integer_value*

  An optional switch and string pair that is identical to the -integer_list_value switch and string pair except that only a single integer is allowed as the value as opposed to a Tcl list of integers.

- -string_value *string_value*

  An optional switch and string pair that is identical to the string_list_value switch and string pair except that only a single string is allowed as the value as opposed to a Tcl list of strings.

- -real_value *real_value*

  An optional switch and string pair that is identical to the real_list_value switch and string pair except that only a single real is allowed as the value as opposed to a Tcl list of reals.

Example 1

The following example defines a simple proc call “example” that displays the arguments it receives. The register_tcl_command defines a command called “example” that accepts a repeatable integer list option, a non-repeatable string option, and a string value. Notice that the registered command will be located in the global namespace and the Tcl proc that it calls is located in the ExampleCommands:: namespace so the two identical names do not collide.

```
namespace eval ExampleCommands {
   proc example {args} {
      puts $args
   }

   register_tcl_command example \
      -string_value {{<chain_name>} {chain}} \
      -string_option { {<label>} {-label} {label} } \
      -integer_list_option { {<range>} {-range} {range*} } \
      -syntax_bnf {<chain_name> [<label>] [<range>]} \
      -context_list {dft patterns} \
      -system_mode_list {analysis} \
      -tcl_proc “[namespace current]::example”
}
```

Example 2

The following example defines a wrapper Tcl proc that is used to call the intercept_with_cell procedure described in of the get_dft_cell command. The code is surrounded in a namespace eval block which puts the intercept_with_cell_wrapper proc in that name space and avoids polluting the global namespace with the Tcl procs when sourcing it. The [namespace current] command, used with the register_tcl_command -tcl_proc option, ensures the tool knows to search in the correct name space. It also ensures that the registered Tcl command in the global namespace does not collide with the Tcl proc of the same name.

After you verify that your registered commands and/or attributes work correctly, you can make them into a *plugin* directory that Tessent Shell automatically loads during initialization. To create the plugin directory, simply place your files in a directory called <plugin>/tcl_modules/<MySpace> where <plugin> and <MySpace> are arbitrary directory names. Set a TESSENT_PLUGIN_PATH environment variable to point to the <plugin> directory and any files with a .tcl extension found inside <plugin>/tcl_modules/<MySpace> will be evaluated into the Tcl namespace ::tcl_modules::<MySpace>. Just like the PATH environment variable, you can specify more than one search location by separating multiple directory names with a colon. A utility command initialize_module is used to load all files ending with .tcl found inside a local directory having the exact same name as the name space.

In the example below, the namespace is called InterceptWithCell which means that initialize_module will source all files matching *./InterceptWithCell/*.tcl*. The advantage of using the initialize_command is that the Tcl proc is only loaded the first time it is needed.

If you name your <plugin> directory name “tessent_plugin” and place it one level above the bin directory of Tessent Shell, the directory will be automatically loaded unless the environment variable TESSENT_PLUGIN_IGNORE_DEFAULT_PATH is defined. Once the plugins are loaded, the command is issued so that they cannot be unregistered accidentally.

```
namespace eval InterceptWithCell {
   proc intercept_with_cell_wrapper {args} {
      initialize_module
      intercept_with_cell_body {*}$args
}
set extra_help "Command accepts a pin, port, or net object as first"
append extra_help "argument. It is intercepted with cell returned by \n"
append extra_help "get_dft_cell <cell_function_name>."

set cell_list "One of mux, clock_mux, and, clock_and, clock_or,"
append cell_list "buffer, clock_buffer, or inverter."

set bnf {<node> <function_name> [<input2_source>] [<select_source>] }
append bnf {[<level>] [<reset_intercept_list>]}

register_tcl_command intercept_with_cell \
   -context_list {dft} \
   -system_mode_list {insertion} \
   -resource_list {design_is_elaborated} \
   -extra_help $extra_help \
   -string_value {"<node>" {node} {} } \
   -string_option [list "<function_name>" "-cell_function_name" \
      {cell_sel} $cell_list {mux clock_mux and clock_and or \
      clock_or buffer clock_buffer inverter}] \
   -string_option {"<input2_source>" "-input2" {input2} {}} \
   -string_option {"<select_source>" "-select" {select} {}} \
   -integer_option {"<level>" "-cascading_level" {level} {} {1 9}} \
   -boolean_option {"<reset_intercept_list>" "-reset_intercept_list" \
      {clear}} \
   -syntax_bnf "$bnf" \
   -visibility hidden \
   -transcript tool_command \
   -tcl_proc "[namespace current]::intercept_with_cell_wrapper master"
}
```

The online help for the registered command is shown here:

```
INSERTION> help intercept_with_cell
   intercept_with_cell <node> -cell_function_name <function_name>
            [-input2 <input2_source>] [-select <select_source>]
            [-cascading_level <level>] [-reset_intercept_list]

DESCRIPTION
   -cell_function_name: One of mux, clock_mux, and, clock_and, or,
clock_or, buffer, clock_buffer

Command accepts a pin, port, or net object as first
argument.It is intercepted with cell returned by
get_dft_cell <cell_function_name>.
```

Related Commands

display_message
lock_current_registration
unregister_tcl_command

remove_from_collection

Context: unspecified, all contexts

Mode: all modes

Usage

remove_from_collection ***base_collection obj_spec***

Description

Removes objects found in obj_spec from a collection, resulting in a new collection.

The base_collection remains unchanged.

If obj_spec is a null collection, the resulting collection is a copy of the base collection. If everything in base_collection is included in obj_spec, the result is an empty collection.

Arguments

- ***base_collection***

  A required value that specifies the base collection from which objects within obj_spec are to be removed.

- ***obj_spec***

  A required value that specifies an object name, a Tcl list of object names, or a collection of objects.

Example

The following example creates a collection of all input and inout ports minus the port called CLK.

set data_in [remove_from_collection [get_ports -direction input inout] CLK]

```
{"in1", "in2"}
```

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
index_collection
is_collection
filter_collection
foreach_in_collection
range_collection
sizeof_collection
sort_collection

rename_instance

Context: all contexts

Mode: insertion

Usage

rename_instance ***obj_spec*** **-new_leaf_name** ***leaf_inst_name*** [-silent]

Description

Renames the leaf name of an instance.

This command returns a collection containing the renamed instance.

If the instance within obj_spec does not exist or if an object already exists with the same leaf_inst_name within the parent_instance, the tool generates an error. You can use the -silent switch to specify to not generate an error message and ignore the renaming request in which case the command returns a null collection.

Arguments

- ***obj_spec***

  A required value that specifies a single instance name or a collection of a single instance object as returned by the get_instance command.

- **-new_leaf_name** ***leaf_inst_name***

  A required switch and value pair that specifies the new leaf name to assign to the instance object referenced in obj_spec.

- -silent

  An optional argument that specifies to ignore any of the two following error conditions: the instance within obj_spec does not exist or if an object already exists with the same leaf_inst_name within the parent_instance.

Example 1

The following example renames the leaf name and3 to my_and3.

```
rename_instance /modB/and3 -new_leaf_name my_and3
```

Example 2

The following example identifies all instances having the user-defined attribute is_my_type set to true and adds a special prefix called “myPrefix_” to their leaf instance name such that it will be easier to find them in the downstream steps of the flow using the SDC command “get_cells myPrefix_* -hierarchical”.

```
foreach_in_collection inst [ get_instance -filter “is_my_type”] {
        set leaf_name [get_attribute_value_list $inst -name leaf_name]
        rename_instance $inst -new_leaf_name myPrefix_${leaf_name}
}
```

Related Commands

create_instance
delete_instances
get_common_parent_instance
replace_instances
uniquify_instances

replace_instances

Context: all contexts

Mode: insertion

Usage

replace_instances ***obj_spec1*** **-with_module** ***obj_spec2*** [-silent]

Description

Replaces the module definition of an instance with a new module definition.

The module definition assigned to the instance during instantiation is replaced with the module specified by the module within obj_spec2.

Connections to pins that exist on both the original and new module remain unchanged. New pins that did not exist on the original module are left open after the replacement. Pins that existed on the original module but do not exist on the new module are silently disconnected.

This command returns a collection containing the new instance.

If obj_spec1 does not exist, the tool generates an error. You can use the -silent switch to specify to not generate an error message when the object within obj_spec1 does not exist. If obj_spec2 is the same module as the original module, no change is made.

Arguments

- ***obj_spec1***

  A required value that is an instance name, a Tcl list of instance names, or a collection of instances as generated by the get_instances command.

- **-with_module** ***obj_spec2***

  A required switch and value pair that specifies the new module object to assign to the instances within obj_spec1. The obj_spec2 value must be a single module name or a collection of a single module object as created by the s command.

- -silent

  An optional argument that specifies to ignore specified objects of obj_spec1 or obj_spec2 that do not exist and to not generate error messages.

Example 1

The following example replaces the module definition of the instance u1/u2 with the module named nor2.

```
replace_instances u1/u2 -with_module nor2
```

Example 2

The following example replaces the module definition for instance u1/u2 to ModB. The original connection to pin u1/u2/a is preserved while the new pin u1/u2/b is left open.

get_attribute_value_list u1/u2 -name module_name

```
{ModA}
```

get_fanin u1/u2/a -stop_on net

```
{u1/n45}
```

replace_instances u1/u2 -with_module ModB

```
{u1/u2}
```

get_fanin u1/u2/a -stop_on net

```
{u1/n45}
```

get_fanin u1/u2/b -stop_on net

```
{}
```

Related Commands

create_instance
delete_instances
get_common_parent_instance
replace_instances
uniquify_instances

report_aborted_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_aborted_faults [*format_type*] [{> | >>} *file_pathname*]

Description

Displays information on testable faults that remain undetected (UD) after the ATPG process.

The report_aborted_faults command can help you determine why the tool aborted faults in the undetected fault list. You can then analyze whether to change the current abort limit to possibly allow ATPG to generate tests for those faults before the tool aborts them. To change the abort limit, use the set_abort_limit command.

Arguments

- *format_type*

  An optional literal that specifies the type of information that you want the tool to display regarding the aborted faults. The literal choices for the *format_type* argument are as follows:

  Summary — A literal that displays a summary of the number of aborted faults for each category. This is the default.

  All — A literal that displays all the aborted faults that are currently in the undetected fault list.

  Backtrack — A literal that displays all the aborted, undetected faults that exceeded the backtrack limit.

  CLock_restriction — A literal that displays all the aborted, undetected faults that had multiple clocks turned on.

  DECisions — A literal that displays all the aborted, undetected faults that exceeded the maximum number of decisions.

  DETected — A literal that displays all the faults that the tool aborted and then later detected.

  Edt — **(Available only when EDT is on)** A literal that displays EDT aborted faults. When the tool generates an effective fault test but is unable to compress the pattern, the fault is classified as an EDT aborted fault. Each of the following increases the probability of EDT aborted faults:

  - Relatively aggressive compression (large chain-to-channel ratio)
  - Large number of ATPG constraints
  - Relatively small design

  INterrupt — A literal that displays all the undetected faults that the tool aborted because you interrupted the ATPG process with a Control-C.

IDdq_restriction — A literal that displays all the undetected faults that the tool aborted while trying to satisfy the IDDQ restrictions.

Ram_sequential — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between the ram_sequential patterns.

SPace — A literal that displays all the aborted, undetected faults that required more memory space than was allocated.

TIme — A literal that displays all the undetected faults that the tool aborted because of the CPU time limitations.

TRansition — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between the initial and final transition pattern.

Write_pass_thru — A literal that displays all the undetected faults that the tool aborted because of inconsistencies between write off and write on for RAM pass through patterns.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example displays the default summary of all the aborted faults:

report_aborted_faults

```
10 backtrack
1  clock_restriction
2  time
```

Related Commands

report_faults
set_abort_limit
set_atpg_limits

report_atpg_constraints

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

Mode: setup, analysis

Usage

report_atpg_constraints[[{> | >>} *file_pathname*] [-Display {DEBug}]]

Description

Displays all the current ATPG state restrictions and the pins on which they reside.

The report_atpg_constraints command displays the pins and their state restrictions defined using the add_atpg_constraints command. The tool uses the state restrictions (constraints) during the ATPG process.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

- -Display DEBug

  An optional switch and literal that specify to display the specific ATPG state of the primary input(s) as callouts in the Debug window.

Example

The following example creates two ATPG pin constraints and then displays the information on those definitions:

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

```
0 /I$135/Q (23)
1 and_b_in
```

Related Commands

add_atpg_constraints

delete_atpg_constraints

report_atpg_functions

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

Mode: setup, analysis

Usage

report_atpg_functions [{> | >>} *file_pathname*]

Description

Displays all the current ATPG function definitions.

The report_atpg_functions command displays the definitions of the ATPG functions created using the add_atpg_functions command. You can use an ATPG function as an argument to the add_atpg_constraints command, which then lets you create state restrictions on pins that the tool uses during the ATPG process.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example creates two ATPG functions and then displays their definitions:

add_atpg_functions and_b_in and /i$143/q /i$141/q /i$142/q
add_atpg_functions select_b_in select /i$144/q /i$142/q
report_atpg_functions

```
USER_AND and_b_in
  Input 0:  /I$143/Q (27)
  Input 1:  /I$141/Q (23)
  Input 2:  /I$142/Q (25)
SELECT   selet_b_in
  Input 0:  /I$144/Q (29)
  Input 1:  /I$142/Q (25)
```

Related Commands

add_atpg_constraints
add_atpg_functions
delete_atpg_functions

report_atpg_timing

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

Mode: setup, analysis

Usage

report_atpg_timing [{> | >>} *file_pathname*]

Description

Reports the current ATPG timing setup.

For complete information, see "Using Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

The report uses the following format:

```
//  ATPG timing                        = { ON | OFF }
//  Timing source                      = { no timing |
                                           SDF minimum delay |
                                           SDF typical delay |
                                           SDF maximum delay }
//  Hold PIs                           = { ON | OFF }
//  Mask POs                           = { ON | OFF }
//  Exclude inter-clock paths          = { ON | OFF ]
//  Timing scale                       = { N/A | 1 ms | 1 us |
                                           1 ns | 1 ps }
//  Slack margin for fault dropping    = { OFf | slack_margin |
                                           slack_margin_percentage% }
//  Delay fault distribution function = { UNIFORM | HIstogram
                                        | EXPONENTIAL | INV_POLY }
//        [ f(t)=function(t), (begin time = min and end time = max ) ]

//  Longest delay (rising, falling)    = ( max_rising, max_falling )
//  Clock waveform                     = { UNDEFINED | USER DEFINED}//
//    Clock Name          Test Clock             System Clock
//                {Period, Offset, Width}   {Period, Offset, Width}
//    ---------------------------------------------------------------
//    clock_name     num1,num2,num3            num1,num2,num3
...
```

- **ATPG timing** — The status of timing-aware ATPG.
- **Timing source** — The source of the timing information.
- **Hold PIs** — The state of holding PIs when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Mask POs** — The state of masking POs when doing static timing analysis for arrival time, propagation time, path delay, and slack.
- **Exclude inter-clock paths** — The state of excluding inter-clock paths when doing static timing analysis for arrival time, propagation time, path delay, and slack.

- **Timing scale** — The time units used. The timing scale is set by the SDF file and defaults to 1 ns.
- **Slack margin for fault dropping** — The margin specified using the set_atpg_timing command.
- **Longest delay** — From static timing analysis.
- **Clock waveform** — The clock information specified by the set_atpg_timing command and reports the following:
  - **Clock Name**
  - **Test Clock** — The ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock). Timing-aware ATPG uses the test clock frequency.
  - **System Clock** — Normal operating frequency.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Examples

The following example shows the timing-aware ATPG settings.

```
read_sdf top_worst.sdf
set_atpg_timing -clock test_clk_dd 16000 8000 8000
set_atpg_timing -clock test_clk_cc 8000 4000 4000
set_atpg_timing -clock test_clk_aa 4000 2000 2000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
report_atpg_timing
```

```
//  ATPG timing                          = ON
//  Timing source                        = SDF typical delay
//  Time scale                           = 1 ns
//  Slack margin for fault dropping      = OFF
//  Delay fault distribution function    = EXPONENTIAL
// f(t) = 0.00158*e^(-0.0021t) + 4.94e-06, (begin time=0, end time=50000)

//  Longest delay (rising, falling)      = (50547.0000, 50547.0000)

//  Clock waveform                       = USER DEFINED
//
//    Clock Name                Test Clock                System Clock
//                        {Period, Offset, Width}   {Period, Offset, Width}
//    ---------------------------------------------------------------------
```

```
//    /test_clk_dd (4)     16000, 8000, 8000        16000, 8000, 8000
//    /test_clk_cc (3)      8000, 4000, 4000         8000, 4000, 4000
//    /test_clk_aa (2)      4000, 2000, 2000         4000, 2000, 2000
//    /clk_in (6)          36000, 18000, 18000      36000, 18000, 18000
```

Related Commands

read_sdf

set_atpg_timing

report_attributes

Context: unspecified, all contexts

Mode: all modes

Usage

Usage 1: Creates a report of attributes defined on a list of objects

report_attributes ***obj_spec*** [-predefined | -user_defined] [-silent] [-verbose]

Usage 2: Creates a report of attributes defined for specific object types

report_attributes **-object_types *type_list*** [-predefined | -user_defined] [-silent] [-verbose]

Usage 3: Creates a report for a specific attribute name

report_attributes **-name *attribute_name*** [-silent] [-verbose]

Usage 4: Creates a report of all registered attributes

report_attributes [-predefined | -user_defined] [-silent] [-verbose]

Description

Creates a human readable report for the set of attributes based on the usage options.

In all usages, specifying the -predefined option restricts the report to only predefined attributes and specifying the -user_defined option restricts the report to only user-defined attributes.

Usage 1: Creates a tabular report for the attributes set on any object found in ***obj_spec.*** The attributes on an object that are inherited from an associated object are reported. For example, an attribute registered for the module object type and set on a given module object is reported in the list of attributes for the instances of that module. By default, attributes which have not been set on the objects are not listed in the report unless the attribute was registered with the -show_default option.

If an object in ***obj_spec*** does not exist, an error message is displayed unless the -silent switch is specified.

Usage 2: Creates a tabular report for all attributes registered on the specified object types. These do not include the inherited attributes from one object type to another such as the module attributed inherited by the associated instances.

Usage3: Creates a tabular detailed report for a specific attribute.

Usage 2: Creates a tabular report for all registered attributes

Arguments

- ***obj_spec***

  A required value used in Usage 1 that specifies an object name, a Tcl list of object names or a collection of objects.

- **-object_types *type_list***

  A required value used in Usage 2 to enumerate the object types for which the attribute report is requested. The currently supported object types are module, port, instance, pin and net. The list of supported object types will be augmented in successive releases and eventually all object types with be available for reporting.

- -name *attribute_name*

  An optional switch and string pair that specify the name of the attribute whose information is to be reported. This argument is mutually exclusive with the obj_spec and -object_types type_list arguments.

- -predefined | -user_defined

  An optional switch that constrains the report to only pre-defined attribute names or only user-defined attribute names. If this argument is not specified, by default no constraints are applied and both pre-defined and user-defined attributes are reported

- -silent

  An optional switch that specifies to suppress error messages if there are objects in the ***obj_spec*** that do not exist.

- -verbose

  An optional switch that specifies to report the full list of possible attribute values. By default, only one line is dedicated to reporting the attribute values; additional values are truncated and replaced with "...".

Example 1

The following example reports the attribute definitions using the report_attributes command:

report_attributes

```
Attribute Definition Report

Attributes defined for all objects:

Name     Object   Type     Class          Default  Constraints
-----------------------------------------------------------------------
name     all      string   Pre-defined    -        -
speed    pin      integer  User-defined   200      100 to 500
added_by net      string   User-defined   John     John Patrick Susan
```

The same result would be achieved by executing:

report_attributes -object_types all

Example 2

The following example reports the attribute definitions for the pin object type:

report_attributes -object_types pin

```
Attribute Definition Report

Attributes defined for 'pin' objects:

Name     Object   Type     Class          Default  Constraints
-----------------------------------------------------------------
name     all      string   Pre-defined    -        -
speed    pin      integer  User-defined   200      100 to 500
```

Example 3

The following example reports the attribute definitions only for a list of pin objects:

report_attributes {u7/CLK u6/A2}

```
Attribute Definition Report

Attributes defined for object 'u7/CLK':
Name                   Value   Inheritance
---------------------  ------  -----------
base_name              u7      -
direction              input   -
is_valid               true    -
leaf_name              CLK     -
module_name            dff     -
name                   u7/CLK  -
object_type            pin     -
parent_instance                -
parent_is_hard_module  true    -

Attributes defined for object 'u6/A2':
Name                   Value  Inheritance
---------------------  -----  -----------
base_name              u6     -
direction              input  -
is_valid               true   -
leaf_name              A2     -
module_name            xor3   -
name                   u6/A2  -
object_type            pin    -
parent_instance               -
parent_is_hard_module  true   -
```

Example 4

The following example reports the attribute definitions for the speed attribute:

report_attributes -name speed

```
Attribute Definition Report

Name: Speed
Description:

    This attribute is used to hold the target frequency that the pin is
    allowed to have.
    The allowed frequency range of 100 to 500 was chosen to match
    the technology spec.

Global Option Settings:
Setting              Value
---------------------------------------------
Class                User-Defined
Type                 int
Default              200
Show_Default         false
Minimum              100
Maximum              500
Object               pin

Specific 'pin' Type settings:
Name                                 Value
---------------------------------------------
dense                                false
export_to_netlist                    off
save_in_flat_model                   off
preserve_boundary_in_flat_model      off
```

Example 5

The following example reports the possible cell_type attribute values with the -verbose switch both enabled and disabled.

report_attribute -name cell_type

```
Attribute Definition Report

Name:
        cell_type

Description:
This is a library model attribute

Setting       Value
------------  -------------------------------------
Class         Pre-Defined
Value_Type    string
Enum          and buffer clock_and clock_buffer ...

GUI_Strings   cell_type = %
Object_Types  module

Specific 'module' Object Type settings:
Name                              Value
--------------------------------  -----
export_during_write               auto
preserve_boundary_in_flat_model   off
applies_to_child_instances        on
display_in_gui                    off
gui_marking_index                 1
```

report_attribute -name cell_type -verbose

```
Attribute Definition Report

Name:
        cell_type

Description:
This is a library model attribute

Setting    Value
---------- ------------------------------------------------------------
Class      Pre-Defined
Value_Type string
Enum       and buffer clock_and clock_buffer
           clock_gating_and clock_gating_or clock_inverter clock_mux clock_or
           dff inverter latch mux nand
           nor or pad scan_cell synchronizer_cell
           xor
GUI_Strings   cell_type = %
Object_Types  module

Specific 'module' Object Type settings:
Name                              Value
--------------------------------  -----
export_during_write               auto
preserve_boundary_in_flat_model   off
applies_to_child_instances        on
display_in_gui                    off
gui_marking_index                 1
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
register_attribute
reset_attribute_value
set_attribute_options
reset_attribute_value
unregister_attribute

report_bist_capture_ranges

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_bist_capture_ranges

Description

Displays a list of the currently defined BIST capture ranges.

The report_bist_capture_ranges command displays a list of the capture ranges that are currently defined as a result of the add_bist_capture_range command. A BIST capture range is defined by a named capture procedure that is stored in the test procedure file, and a range of patterns that will use this capture procedure. The complete range of patterns reported by the report_bist_capture_ranges command typically represents a small subset of the entire pseudo-random pattern space, and is applied to the complete pattern space by repeating the sequence until all the pseudo-random patterns have been simulated.

Example

The following example shows the output from the report_bist_capture_ranges command.

report_bist_capture_ranges

```
// List of currently dedined Logic BIST capture ranges
// Procedure name  from pattern  to pattern
// --------------  ------------  ----------
// cap1                0           14
// cap2               15           15
//
// NOTE: This list defines the active named capture range during pseudo-
// random pattern simulation. The tool automatically restarts at the top
// of the list when the pattern counter matches the final "to pattern"
// value.
```

Related Commands

add_bist_capture_range

delete_bist_capture_ranges

report_black_boxes

Context: all contexts

Mode: all modes

Usage

report_black_boxes {{**-Instances** [*ins_pathnames*]} | {**-Modules** [*module_names*]} | **-All** | **-Undefined**} [{> | >>} *file_pathname*]

Description

Displays information on blackboxes and undefined models.

The report_black_boxes command reports on the status of any instance- or module-based blackboxes, or undefined models that are not yet blackboxed. The report follows the following format.

```
MODULE: module_name (default tie value = 0|1|X|Z)
  SYSTEM: Inout|Output pin pin_name tied to 0|1|X|Z
  USER: Inout|Output pin pin_name tied to 0|1|X|Z
INSTANCE: ins_name (default tie value = 0|1|X|Z)
  SYSTEM: Inout|Output pin pin_name tied to 0|1|X|Z
  USER: Inout|Output pin pin_name tied to 0|1|X|Z
```

For module-based blackboxes, the tool displays the string MODULE followed by the name of the module and the default tie value (0, 1, X, or Z). The tool then displays a list of module pins. For each pin, the tool displays either SYSTEM or USER followed by the direction type of the pin (Inout or Output), the name of the pin, and its tied value. SYSTEM declares that the pin is tied to the default value by the system while USER declares that you explicitly tied the pin to the specified value.

For instance-based blackboxes, the report replaces the string MODULE with INSTANCE to explicitly declare that it is an instance-based blackbox.

Arguments

- **-Instances** [*ins_pathnames*]

  A switch and optional string that specify for the tool to display information on instance-based blackboxes. The ins_pathnames argument can be an instance, a Tcl list of instances, or a collection of instances. If you do not specifiy an ins_pathname, the tool displays information on all instance-based blackboxes. If you specify an instance pathname, it reports on that single, instance-based blackbox.

- **-Modules** [*module_names*]

  A switch and optional string that specify for the tool to display information on module-based blackboxes. The module_names argument can be an instance of the Verilog module or ATPG library model, a Tcl list of instances, or a collection of instances. If you do not supply a module_name, the tool displays information on all module-based blackboxes. If you specify a *module_name*, it reports on that single, module-based blackbox.

- **-All**

  A switch that specifies for the tool to display information on all defined blackboxes, as well as undefined models. This is the command default.

- **-Undefined**

  A switch that specifies for the tool to display information on undefined models that have not yet been blackboxed. Use this switch to determine whether your design is complete, or is missing library models. If you intend to blackbox undefined models, this report allows you to verify that only the intended models are undefined.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Example

The following example defines module- and instance- based blackboxes then reports on them.

add_black_boxes -module core -pin do1 0 -pin io1 1
add_black_boxes -instance core1 1 -pin do0 0 -pin io0 0
report_black_boxes -all

```
Default tie value = X
MODULE: core
SYSTEM: Output pin do0 tied to X
  USER: Output pin do1 tied to 0
  SYSTEM: Inout pin io0 tied to X
  USER: Inout pin io1 tied to 1
INSTANCE: core1
  USER: Output pin do0 tied to 0
  SYSTEM: Output pin do1 tied to 1
  USER: Inout pin io0 tied to 0
  SYSTEM: Inout pin io1 tied to 1
```

Related Commands

add_black_boxes
delete_black_boxes
report_tied_signals

report_bus_data

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

Mode: setup, analysis

Usage

report_bus_data ***type*** [{> | >>} *file_pathname*]

Description

Displays the bus data information for either an individual bus gate or for the buses of a specific type.

The report_bus_data command displays the following bus information:

- Instance name
- Gate identification number
- Contention handling (pass, bidi, fail, or abort)
- Type of bus (strong or weak)
- Number of drivers on the bus
- Any learned behavior of the bus.

The design rule that checks for bus contention mutual exclusivity is rule E10. For more information on rule E10, refer to the "Extra Rules (E Rules)" section.

If you enable the learn reporting by using the set_learn_report command, the tool provides the following two additional lines of information with the report_bus_data command:

- Information on whether or not the bus is capable of being set to an X, set to a Z, or having multiple drivers turned on.
- A list of drivers and their corresponding gate type. Drivers that are equivalent have a gate type of EQ.

The design rule that checks to see if there is any possible input combination that can force a bus into the high-impedance state (Z) is rule E11. For more information on rule E11, refer to the "Extra Rules (E Rules)" section.

Arguments

- ***type***

  A required literal or integer that specifies the type of bus for which you want the tool to display information. The choices for the *type* argument are as follows:

  gate_id# — An integer that specifies the gate identification number whose bus data you want to display.

  ALl — A literal that displays the bus data for all buses.

Weak — A literal that displays the bus data for the weak buses.

Strong — A literal that displays the bus data for the strong buses.

Dominant — A literal that displays the bus data for the final bus of every set of cascaded buses.

NONDominant — A literal that displays the bus data for all but the final bus in every set of cascaded buses.

Pass — A literal that displays the bus data for the buses that passed the contention mutual exclusivity checking.

Bidi — A literal that displays the bus data for the bidirectional buses that have possible contention problems. For the tool to place a bus in this category, the bidirectional pin must have only a single tri-state driver.

Fail — A literal that displays the bus data for the buses that failed the contention mutual exclusivity checking.

ABort — A literal that displays the bus data for the buses that aborted contention mutual exclusivity checking.

Buf — A literal that displays the bus data for all buses that have the buffer learned behavior.

Xor — A literal that displays the bus data for all buses that have the exclusive OR learned behavior.

Mux — A literal that displays the bus data for all buses that have the multiplexer learned behavior.

AND — A literal that displays the bus data for all buses that have the AND learned behavior.

OR — A literal that displays the bus data for all buses that have the OR learned behavior.

POSS_Mult_dr_on — A literal that displays the bus data for all buses that have a possibility of having multiple drivers turned on at the same time.

POSS_X — A literal that displays the bus data for all buses that have a possibility of being at an unknown state.

POSS_Z — A literal that displays the bus data for all buses that have a possibility of being at the high-impedance state.

HIstogram — A literal that displays a summary of information identifying the number of buses that are in each of the possible categories.

ZPass — A literal that displays the bus data for the buses that pass the E11 design rule.

ZFail — A literal that displays the bus data for the buses that fail the E11 design rule check.

ZAbort — A literal that displays the bus data for the buses that abort the E11 design rule check.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example displays the information on a specific bus gate—an inverter (INV):

report_bus_data 31

```
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
   Bus Drivers:  30(SW) 28(SW)
```

Example 2

The following example first enables access to the static learning data, then displays both the learned information and the bus data on a specific bus gate, again, an inverter (INV):

set_learn_report on
report_bus_data 31

```
/FA1/XOR1/OUT/ (31)Handling=pass type=strong #Drivers=2 (INV)
   Learn Data :  poss_X=no, poss_Z=no, poss_mult_drivers_on=no
   Bus Drivers:  30(SW) 28(SW)
```

Related Commands

set_learn_report

report_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_bypass_chains
[-All_blocks | {-Edt_chain *chain_name*} | {-Bypass_chain *chain_number*}]
[{> | >>} *file_pathname*]

Description

Displays the defined EDT bypass scan chain connections.

If issued from Setup mode, the report_bypass_chains command lists scan chain connections for bypass scan chains defined using the set_bypass_chains command.

If issued from analysis mode, all bypass scan chain connections are reported, whether defined by Tessent TestKompress or with the set_bypass_chains command.

You can display the chain connection information for all bypass chains, or optionally include the -Edt_chain or -Bypass_chain switch to display only the information for a particular bypass chain. The order, left to right, in which the display lists the scan chains in a particular bypass chain is the order in which the chains are series-connected from channel input to channel output.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT bypass chains in every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “Modular Compressed ATPG” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-**Edt_chain *chain_name*

  An optional switch and string pair that specify to display the scan chain connections for only the EDT bypass chain that includes the EDT scan chain, *chain_name*. You can use this switch to see which bypass chain or scan channel is associated with a particular scan chain.

- **-**Bypass_chain *chain_number*

  An optional switch and positive integer pair that specify to display the scan chain connections for the EDT bypass chain, *chain_number*. Use this switch to list the scan chain connections for a particular EDT bypass chain. The *chain_number* is the same as the number of the scan channel connected to the bypass chain.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

Assume a design has eight scan chains named “chain1” through “chain8” and that the chains have been incorporated into two EDT bypass channels configured from two EDT scan channels. The following example displays the scan chain connection information for the EDT bypass chain containing chain3. You can see from the report that chain3 is part of a bypass chain connected to scan channel 2.

report_bypass_chains -edt_chain chain3

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ------
//       2     chain2, chain3, chain4, chain7, chain8
```

Example 2

The following example displays the connection information for EDT bypass chain 1 (scan channel 1).

report_bypass_chains -bypass_chain 1

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ------
//       1     chain1, chain5, chain6
```

Again, notice that the EDT bypass chain number (Bypass# in the display) is the same as the number of the scan channel to which the bypass chain is connected.

Related Commands

reset_bypass_chains

set_bypass_chains

report_capture_handling

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

Mode: setup, analysis

Usage

report_capture_handling [List | SUmmary | SOurces | SInks | Gates]
[{> | >>} *file_pathname*]

Description

Displays any special data capture handling currently in use.

To change how the tool handles data capturing on level-sensitive and trailing-edge state elements, use the add_capture_handling command. If you have not set up any special data capture handling, then the report_capture_handling command does not generate a report.

Arguments

- List

  A literal that displays the handling settings (old, new, or X), whether the element is a sink or source, the instance pathname, and the gate identification number. This is the default.

- SUmmary

  A literal that displays the handling settings for both level-sensitive and trailing-edge state elements, the number of sources, the number of primitive gates in the flattened netlist between source and sink points, and the number of sinks.

- SOurces

  A literal that displays the gates that have source-point, special data capture handling.

- SInks

  A literal that displays the gates that have sink-point, special data capture handling.

- Gates

  A literal that displays the identification numbers of all the primitive gates between the source and sink points.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example sets up the data capture handling for one gate, and then issues the set_capture_handling command to identify the associated sinks for that source:

add_capture_handling new 1158 -source
set_capture_handling

The following set of commands show the different formats for the available reports:

report_capture_handling

```
NEW Source /I_1_16/DF0/ (1158)
```

report_capture_handling list

```
NEW Source /I_1_16/DF0/ (1158)
```

report_capture_handling summary

```
Capture handling summary: LS=OLD, TE=NEW, #sources=1,
                                          #int_gates=14, #sinks=2
```

report_capture_handling source

```
Source list : 1158-X
```

report_capture_handling sink

```
Sink list : 1160-X 1165-X
```

report_capture_handling gates

```
Int_gate list: 13 14 39 40 51 52 53 61 62 63 68 69 79 86
```

Related Commands

add_capture_handling

set_capture_handling

report_capture_procedures

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

Mode: setup, analysis

Usage

report_capture_procedures [-CLock_sequence]
[-CReated_capture_procedure | -Loaded_capture_procedure | -ENabled | -Disabled | *capture_procedure_name*...] [-All | -Summary | -Internal | -External] [-Display Wave] [{> | >>} *file_pathname*]

Description

Displays the cyclized information for a specified named capture procedure.

Cyclizing is a processing step performed prior to ATPG that merges internal mode cycles when needed to optimize them and/or expand other internal mode cycles to ensure correct simulation results.

For example, when a named capture procedure contains a cycle that pulses multiple clocks at different times, the cycle must be expanded into multiple cycles (one for each clock pulse) in order to simulate it. The cyclized information shows the cycles ATPG will actually use when creating patterns, after any merging or expansion of cycles. The display also includes comments about the timing of the cycles and where the at-speed sequences begin and end to aid your understanding.

The displayed information resembles the original procedure as defined in the test procedure file, but it does not have any timeplates. Therefore, it will not work if copied and pasted as is into a dofile; you must add timeplates. To display the procedure as it actually exists in the test procedure file, use the report_procedures command.

The number of cycles reported is the sequential depth used during test generation. The display of merged or expanded cycles is illustrated in the examples. See also "Displaying Named Capture Procedures" in the *Tessent Scan and ATPG User's Manual*.

Note

If the named capture procedure is modified to cyclize it for ATPG, the modifications are internal only; the named capture procedure is not altered in the test procedure file.

Arguments

- -Clock_sequence

  An optional switch that displays the clock sequence of the named capture procedures. The scan load cycle(s), at-speed cycle(s), and the number of patterns the named capture procedure is associated with are also displayed.

- -Created_capture_procedure

  An optional switch that displays only the created named capture procedures.

- -Loaded_capture_procedure

  An optional switch that displays only the named capture procedures loaded from the read_procfile command or from the procedure file associated with the scan group definition.

- -Enabled

  An optional switch that displays only enabled named capture procedures.

- -Disabled

  An optional switch that displays only disabled named capture procedures.

- *capture_procedure_name*

  An optional, repeatable string that displays only the specified named capture procedures. By default, information for all named capture procedures is displayed.

- -All

  An optional switch that displays information for both the internal and external modes of the named capture procedure. This is the default.

- -Summary

  An optional switch that displays only summary information about the named capture procedure.

- -Internal

  An optional switch that displays information for only the internal mode of the named capture procedure.

- -External

  An optional switch that displays information for only the external mode of the named capture procedure.

- -Display Wave

  An optional switch that displays the specified capture procedure in the Wave window of DFTVisualizer. This is for optimized NCP's. Use the report_procedures command to display unoptimized NCP's.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to create or replace the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append the contents of *file_pathname*.

Example 1

The following example displays summary information for the named capture procedures *nam_cap1*, *nam_cap2*, *my_cap1*, and *my_cap2*, including the scan load cycle(s), the at-speed cycle(s), and the number of patterns with which the named capture procedures are associated. It

also indicates *nam_cap1* and *nam_cap2* were loaded from an external file and *my_cap1* and *my_cap2* were internally created.

report_capture_procedures -summary

```
capture_proc_name  cycles  at_speed_cycle  active  source   # associated patterns
                                                             internal  external
-----------------  ------  --------------  ------  -------  ---------  ---------
nam_cap1             1          1          no     loaded        0         -
nam_cap2             1          1          yes    loaded       10         -
my_cap1              2        {1 to 2}     yes    created       5         -
my_cap2              2        {2 to 2}     no     created       0         -
```

Example 2

The following example displays the clock sequence of the named capture procedures *capture1*, *capture2*, *user_capture_proc*, and *user_capture_proc_1* in addition to the summary information.

report_capture_procedures -clock_sequence

```
capture_proc_name  cycles  at_speed_cycle  active  source   # associated patterns  capture_clock_sequence
                                                             internal   external
-----------------  ------  --------------  ------  -------  ----------  ----------  ----------------------
capture1             1          1          yes    loaded        6           -      S(c4,c3,c2,c1,c0)
capture2             1          1          yes    loaded        8           -      S(c9,c8,c7,c6,c5)
user_capture_proc    3       {1 to 3}      yes    created       0           -      S(c0)(c1,c0)(c3,c2)
user_capture_proc_1  3       {1 to 3}      yes    created       0           -      S(c0),(c1,c0)(c3,c2)
```

Example 3

The following example displays information for all the named capture procedures (*capture1*, *capture2*, and *my_cap1*). The first line displayed for each named capture procedure indicates if the procedure is currently enabled for test generation (ACTIVE) or disabled (INACTIVE). You can enable or disable capture procedures with the set_capture_procedures command.

report_capture_procedures

```
procedure capture "capture1" = //  (INACTIVE) (loaded capture procedure)
    cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure
PO)
      condition  "/inst_0/clkenhold/reg_sco/"  1 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[1]" ;
      pulse "/clock[0]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "capture1"

procedure capture "capture2" = //  (ACTIVE) (loaded capture procedure)
```

```
   cycle = // cycle 1 starts at time 0 (Begin_AC_test) (Force PI) (Measure
PO)
      condition  "/inst_0/clkenhold/reg_sco/"  0 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[4]" ;
      pulse "/clock[3]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "capture2"

procedure capture "my_cap1" = //  (ACTIVE) (created capture procedure)
    cycle = slow // cycle 1 starts at time 0 (Force PI)
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[4]" ;
      pulse "/clock[3]" ;
    end ; // cycle 1
    cycle = // cycle 2 starts at time 50 (Begin_AC_test) (Force PI)
(Measure PO)
      condition  "/inst_0/clkenhold/reg_sco/"  0 ;
      force  "/clock[4]"  0 ;
      force  "/clock[3]"  0 ;
      force  "/clock[2]"  0 ;
      force  "/clock[1]"  0 ;
      force  "/clock[0]"  0 ;
      force  "/clear"  0 ;
      force  "/scan_en"  0 ;
      force  "/test_clk"  0 ;
      force_pi ;
      measure_po ;
      pulse "/clock[2]" ;
      pulse "/clock[1]" ;
    end ; // cycle 1 (End_AC_test)
    // mode external is not defined.
end ; // procedure "my_cap1"
```

Example 4

The following example displays how the tool will use the cycle during ATPG (the command's output is excerpted for brevity):

report_capture_procedures -internal

```
...
   Mode internal =
      Cycle =   // cycle 1 starts at time 0 (Begin_AC_test)
                // (Force PI)
         Force_pi ;
         Pulse /clk1 ;
      End ;    // cycle 1 at time 20
      Cycle =  // cycle 2 starts at time 20
         Pulse /clk2 ;
      End ;    // cycle 2 at time 50
...
```

The display shows that for this particular design and cycle definition, the tool expands the one cycle from the test procedure file into two cycles internally. Notice that the combined time of the two cycles is the same as the period of the original cycle in the test procedure file. The tool will automatically adjust the sequential depth used during ATPG to accord with the resultant change in number of cycles.

Example 5

The next example shows a case where the tool merges two cycles described in the test procedure file into one cycle for ATPG. The relevant excerpts from the test procedure file are shown first; the example then displays the cyclized data for the three cycles.

```
mode_internal =
...
   timeplate tp2;
   cycle =
      force_pi;
      force tms 0;
      pulse clk;
   end;
   cycle =
      force tms 0;
   end;
   cycle =
      force tms 0;
      pulse clk;
   end;
...
```

```
timeplate tp2 =
   force_pi 0;
   pulse clk 10 10;
   period 50;
end;
```

report_capture_procedures -internal

```
...
   Mode internal =
      Cycle =  // cycle 1 starts at time 0 (Begin_AC_test)
         // (Force PI)
         Force_pi ;
         Force /tms 0 ;
```

```
   Pulse /clk ;
End ;    // cycle 1 at time 100
Cycle =  // cycle 2 starts at time 100
   Force /tms 0 ;
   Pulse /clk ;
End ;    // cycle 2 at time 150
```

Related Commands

create_capture_procedures
report_procedures
report_timeplate
set_capture_procedures

report_cell_constraints

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

Mode: setup, analysis

Usage

report_cell_constraints [-Scan_cell | -Non_scan_cell]
[{-CHain *chain_name*} | {-CLock *clock_name*}]
[{> | >>} *file_pathname*]

Description

Displays a list of the constrained cells.

The report_cell_constraints command displays a list of all the scan cells (also non-scan cells for DX and SX constraints) that you previously constrained to a constant value using the add_cell_constraints command. The display may consist of up to seven columns, as follows (columns that contain no data for all cell constraints are not displayed):

- Constraint value — Specifies the constraint value.
- Chain name — Specifies the scan chain name if the constraint is on a scan cell. If the constraint is on a non-scan cell, then this column is blank.
- Cell position — Specifies the position in the scan chain if the constraint is on a scan cell. If the constraint is on a non-scan cell, then this column is blank.
- Cell instance name — Specifies the cell instance of the constraint. This column is blank unless you originally specified the constraint with a *pin_pathname*.
- Gate instance name — Specifies the gate instance of the constraint. This column is blank unless you originally placed the constraint on a non-scan cell using the -Clock switch.
- Constraint properties — Displays constraint properties, as applicable, from the following list:.
  - Internal — Constraint was applied by the tool automatically in response to certain DRC results
  - Whole Chain — Constraint was applied using the -Chain switch
  - Drc C6 — Constraint was applied using the “-Drc C6” argument combination.
  - Clock — Constraint was applied using the -Chain switch
  - Drc D8 Warn — Constraint was applied by the tool automatically to a D8 failing master cell when “set_drc_handling d8 warning” was in effect for DRC. See “D8” for more information.
  - Dynamic — Constraint was applied in Analysis mode
  - Static — Constraint was applied in Setup mode

- Pattern scope — Specifies the pattern scope of the constraint: whether the tool uses the constraint only when creating scan patterns, or uses the constraint for both scan and chain pattern creation.

Arguments

- -Scan_cell

  An optional switch that specifies to report_cell_constraints at scan cells only. This switch is valid in Analysis mode only.

- -Non_scan_cell

  An optional switch that specifies to report_cell_constraints at non-scan cells only. This switch is valid in Analysis mode only.

- -CHain *chain_name*

  An optional switch and string pair that specifies to report_cell_constraints added to the scan chain, *chain name*, by an earlier add_cell_constraints -Chain command. This switch is valid in Analysis mode only.

- -CLock *clock_name*

  An optional switch and string pair that specifies to report the cell constraints added by an earlier add_cell_constraints -Clock command to latches and flip-flops controlled by the clock, *clock_name*. This switch is valid in Analysis mode only.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example begins in Setup mode and constrains a single scan cell to be at a constant zero for the generation of all patterns. The example then displays current cell constraints, leaves Setup mode, triggering the tool's automatic DRC process and again reports the cell constraints:

add_cell_constraints sdff1/q c0 -all_patterns
report_cell_constraints

```
// constraint cell_instance  constraint     pattern scope
//    value       name       properties
// ---------- -------------  ---------- ---------------------
//     C0        sdff1/q      (Static)  Chain + Scan Patterns
```

set_system_mode analysis
report_cell_constraints

```
// const chain  cell    cell_inst constraint     pattern scope
```

```
// value  name  pos        name   properties
// ----- -----  -----  ---------- ---------- ---------------------
//   C0    c1    2        sdff1/q  (Static)  Chain + Scan Patterns
```

Notice that once the tool has identified the scan chains during the DRC process, the report identifies the chain and cell position of the constrained scan cell.

Example 2

The next example adds DX constraints on all flip-flops clocked by clk2. You can see from the cell constraint report that these flip-flops are not scan cells (no chain or cell position). Notice also that the gate instances are reported for them since each flip-flop is a standalone gate, not part of a scan cell.

add_cell_constraints -clock /clk2 -type flop dx
report_cell_constraints

```
// const chain  cell  cell_inst gate_inst constraint     pattern scope
// value  name  pos      name     name    properties
// ----- ----- ------ --------- --------- ---------- --------------------
//   DX                          /dff6   (Clock)    Scan Patterns only
//                                       (Dynamic)
//   DX                          /dff5   (Clock)    Scan Patterns only
//                                       (Dynamic)
//   C0    c1    2      sdff1/q          (Static)   Chain + Scan Pattern
```

Related Commands

add_cell_constraints

delete_cell_constraints

report_cell_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_cell_models [-All | {-Type *cell_model_type*}]

Description

Displays a list of either all cell models or the DFT library models associated with the specified cell type.

The report_cell_models command displays the cell models that you either added with the add_cell_models command, or described in the DFT library with the cell_type attribute.

Arguments

- -All

  An optional switch that specifies to display all cell model definitions that you added with the add_cell_models command. This is the default.

- -Type *cell_model_type*

  An optional switch and literal pair that specifies to display a listing of all the cell models of a particular type. The valid cell_model_types are as follows (with the minimum typing characters shown in uppercase):

  INV — A literal that specifies a one-input inverter gate.

  And — A literal that specifies a two-input AND gate.

  Buf — A literal that specifies a one-input buffer gate.

  OR — A literal that specifies a two-input OR gate.

  NAnd — A literal that specifies a two-input NAND gate.

  NOr — A literal that specifies a two-input NOR gate.

  Xor — A literal that specifies a two-input exclusive OR gate.

  Mux — A literal that specifies a 2-1 multiplexer.

  Scancell — A literal that specifies a cell with four input pins (clock, data, scan in, and scan enable), clocked scan cell with four inputs (clock, data, scan clock, and scan enable), or LSSD scan cell with five inputs (clock, data, scan in, master clock, and slave clock).

  DFf — A literal that specifies a D flip-flop with two input pins (clock and data).

  LATCH — A literal that specifies a D latch with two input pins (enable and data).

Examples

The following example displays a list of all added cell models:

```
add_clocks 0 clk
set_test_logic -set on -re on -clock on
set_system_mode analysis
report_dft_check
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Commands

add_cell_models
delete_cell_models
set_test_logic

report_chain_masks

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

Mode: setup, analysis

Usage

report_chain_masks

Description

Reports the list of masked scan chains and their associated unload value.

Arguments

None

Example

The following example adds several chain masks and then reports the added chain masks:

add_chain_masks chain2 chain4 chain5
report_chain_masks

```
// Masked scan chain Unload value Cell constraints
// ----------------- ------------ ----------------
//      chain2             X              OX
//      chain4             X              OX
//      chain5             X              OX
```

The following example adds several chain masks, specifies load values, and then reports the added chain masks:

add_chain_masks chain2 -load_value 0
add_chain_masks chain4 -load_value 0
add_chain_masks chain5 -load_value 1
report_chain_masks

```
// Masked scan chain Load value Unload value Cell constraints
// ----------------- ---------- ------------ ----------------
//      chain2           0            X              OX
//      chain4           0            X              OX
//      chain5           1            X              OX
```

Related Commands

add_chain_masks

delete_chain_masks

report_clock_controller_pins

Context: dft -logic_bist

Mode: setup, analysis

Usage

report_clock_controller_pins [> | >> *file_pathname*]

Description

Reports the clock controller pins specified with the set_clock_controller_pins command.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

This example reports all clock_controller pins specified so far using the set_clock_controller_pins command.

report_clock_controller_pins

```
//
//  PinType                  Connection
//  -------                  ----------
//  Shift clock              clock_control_i1/SLOW_CLK
clock_control_i2/SLOW_CLK clock_control_i3/SLOW_CLK
//  Scan enable              clock_control_i1/SCAN_EN
clock_control_i2/SCAN_EN clock_control_i3/SCAN_EN
//  LBIST enable            clock_control_i1/LBIST clock_control_i2/LBIST
clock_control_i3/LBIST
//  Capture procedure index  NCP_INDEX_1/A NCP_INDEX_0/A
//
```

Related Commands

set_clock_controller_pins

set_lbist_power_controller_options

report_clock_controls

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

Mode: setup, analysis

Usage

report_clock_controls

Description

Displays current configuration information for clock control definitions.

For more information, see "Support for Internal Clock Control" in the *Tessent Scan and ATPG User's Manual*.

Example

The following example illustrates how clock control definitions are reported.

The following clock control is defined in a test procedure file.

```
clock_control clk =
   source_clock pll_clk;
   atpg_cycle 0=
     condition ctl_dff2/Q 1;
   end;
   atpg_cycle 1=
     condition ctl_dff1/Q 1;
   end;
end;
```

The Report Clock Control command reports the definition as follows:

report_clock_controls

```
CLOCK_CONTROL "/clk (6)" =
SOURCE_CLOCK "/pll_clk (4)";
   ATPG_CYCLE 0 =
   CONDITION "/ctl_dff2/ (56)" 1;
  END;
   ATPG_CYCLE 1 =
   CONDITION "/ctl_dff1/ (55)" 1;
  END;
END;
```

Related Commands

set_clock_controls

report_clock_domains

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_clock_domains [{-Compatible_clocks [-Details]} | {-Race_points *clk1 clk2*}] [{> | >>} *file_pathname*]

Description

Displays all clocks and their corresponding clock domain numbers.

During ATPG, if “set_clock_restriction domain_clock” is set, the tool may simultaneously turn on any subset of compatible clocks. This allows fault effects to be captured by more than one clock in the same cycle without risk of simulation mismatches due to clock skew. As a result, the test pattern count may be reduced and the fault coverage improved for the same clock sequential depth, compared with when only one clock is turned on in each cycle.

The report_clock_domains command lists defined clocks by groups based on their compatibility. Clocks are compatible (occupy the same clock domain) if capturing with them simultaneously can be performed reliably. Use this command to assess the overall clock compatibilities in your design, and to determine if using “set_clock_restriction domain_clock” would be beneficial.

Note

The basic display enables you to quickly evaluate the overall clock compatibility of your design. It is summary information, however, and does not necessarily show the compatibility relationship between all possible clock pairings. That is, two clocks listed with different clock domains are not necessarily incompatible. Use the -Compatible_clocks switch if you need to check the compatibility information for a specific pair of clocks.

Arguments

- -Compatible_clocks

  An optional switch that includes in the display the pair-wise clock compatibility information the tool uses during ATPG. Because two clocks with different clock domain numbers are not necessarily *in*compatible, this switch enables you to determine precisely whether two clocks are compatible or not.

  The tool treats two clocks as compatible if the clocks can be in their “on” state at the same time without causing clock skew problems or race conditions. To make this determination, the tool analyzes the clocking relationship between source and sink memory elements where the source is clocked by one clock and the sink is clocked by a different clock.

Table 7-22 summarizes the conditions under which the tool determines the two clocks to be compatible (C) or incompatible (I).

Table 7-22. Clock Compatibility Summary

Source	Sink			
	LE[1]	TE[2]	AH[3](TE)	AL[4](LE)
LE	I	C	C	I
TE	C	I	I	C
AH (LE)	I	C	C	I
AL (TE)	C	I	I	C

1. LE = Captures on PI clock's leading edge
2. TE = Captures on PI clock's trailing edge
3. AH = Active high, level sensitive element
4. AL = Active low, level sensitive element

- -Details

  An optional switch that displays the same compatibility information provided by the -Compatible_clocks switch, but instead of using an "x" in the display to mark each incompatible clock pair, use as the marker the number of state elements at which the tool determined the pair to be incompatible.

- -Race_points *clk1 clk2*

  An optional switch and pair of strings that lists the instances that cause the two clocks, *clk1* and *clk2*, to be incompatible. The tool includes the gate ID and type of state element (scan, non-scan or RAM) for each instance listed.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list or creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname*.

Example 1

The following example displays the defined clocks and their domains of the design. The numbers in parentheses are the unique gate identification numbers assigned to each gate during the model flattening process.

set_system_mode analysis
report_clock_domains

```
// Clock name   Domain
// -------------------
// /ck11m (8)        1
```

```
// /ck22m (9)         1
// /ck2_75m (10)      2
// /ck5_5m (11)       2
// /nreset (39)       3
// /scan_reset (63)   4
//
// 6 clocks classified into 4 domains.
// This is only one of the possible clock classifications. To see all
dependencies of clock pairs, use 'report_clock_domains -Compatible_clocks'
```

Example 2

The following example displays the clock domains in example 1 with added information about the possible clock pairs in the design. Notice that although the tool assigned the /scan_reset clock to a domain different from the others, it is actually incompatible only with clocks /ck2_75m, and /nreset. This is indicated by an "x" in the /scan_reset row, above the index number that corresponds to each of those two clocks.

report_clock_domains -compatible_clocks

```
// No:Clock name  Domain | Clock compatibility
// ----------------------+---------------------
// 1: /ck11m (8)        1| .
// 2: /ck22m (9)        1| ..
// 3: /ck2_75m (10)     2| .x.
// 4: /ck5_5m (11)      2| ....
// 5: /nreset (39)      3| .xxx.
// 6: /scan_reset (63)  4| ..x.x.
// ----------------------+---------------------
//                    No.| 123456
// . = Compatible clock pair
// x = Incompatible clock pair
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

Example 3

The following example displays the same information as the preceding example, but directs the tool to substitute for each "x" the number of state elements at which the tool found the pair to be incompatible.

report_clock_domains -compatible_clocks -details

```
// No:Clock name  Domain | Clock compatibility
// ----------------------+---------------------
// 1: /ck11m (8)        1| .
// 2: /ck22m (9)        1| . .
// 3: /ck2_75m (10)     2| . 3 .
// 4: /ck5_5m (11)      2| . . . .
// 5: /nreset (39)      3| . 6 1 6 .
// 6: /scan_reset (63)  4| . . 8 . 4 .
// ----------------------+---------------------
//                    No.| 1 2 3 4 5 6
//        . = Compatible clock pair
// <number> = Incompatible clock pair
```

```
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
```

The report shows there are three state elements that cause clocks /ck2_75m and /ck22m to be incompatible. The next example lists the instance name, gate ID and type of element for each of these elements.

report_clock_domains -race_points /ck2_75m /ck22m

```
//    /ff30/ (57) non-scan
//    /ff31/ (58) non-scan
//    /ff31b/ (58) non-scan
//  3 state elements cause clocks "/ck2_75m" and "/ck22m" to be
//  incompatible.
```

Example 4

The following example uses the add_norace_clocks command to define three clocks as compatible even though the analysis has determined two of the three possible pairings from among the three clocks are incompatible. The example then reports on the compatibility of clock pairs. In the resultant report, you can see that an “n” is used in place of the “.” if the add_norace_clocks command was applied to a clock pair that the tool already determined was compatible; and an “N” is used if the norace definition was applied to an incompatible pair.

add_norace_clocks /ck22m /ck5_5m /nreset
report_clock_domains -compatible_clocks

```
// No: Clock name  Domain | Clock compatibility
// -----------------------+--------------------
// 1: /ck11m (8)         1| .
// 2: /ck22m (9)         1| ..
// 3: /ck2_75m (10)      2| .x.
// 4: /ck5_5m (11)       2| .n..
// 5: /nreset (39)       3| .NxN.
// 6: /scan_reset (63)   4| ..x.x.
// -----------------------+--------------------
//                     No.| 123456
// . = Compatible clock pair
// x = Incompatible clock pair
// n = User-defined norace between compatible clk pair (1)
// N = User-defined norace between incompatible clk pair (2)
//
// 6 clocks classified into 4 domains.
// This is one of possible clock classifications.
//
// Warning: 2 incompatible clock pair(s) overwritten by the user as norace
clock pair(s).
```

Note

The tool assumes you have expert knowledge when you override its analysis, but will remind you of the override with a warning.

Related Commands

add_clocks
add_norace_clocks
delete_clocks
set_clock_restriction

report_clock_gating

Context: dft -scan, dft -test_points

Mode: analysis

Usage

report_clock_gating [-LIbrary_model *library_model_name*…]
[-Module *module_object*...] [-INStance *instance_object*...]
[{> | >>} *file_pathname*]

Description

Reports the clock gating instances that were identified as having unconnected ports and were connected to either the scan enable signal or to a user-specified signal or, reports the unconnected ports of the specified clock gating cells only.

Arguments

- -LIbrary_model *library_model_name*

  An optional switch and a repeatable string that specify the library model name(s) of the clock gating cell instances to be reported for unconnected ports to be connected to either the scan enable signal or the specified signals.

- -Module *module_object*

  An optional switch and string pair that specify the netlist module name(s) of the clock gating cell instances to be reported for unconnected ports to be connected to the specified signals. The module_object can be a single module name, a TCL list of module names, or a collection of module objects as returned by the get_modules command.

- -INstance *instance_object*

  An optional switch and string pair that specify the hierarchical pathname(s) of the clock gating cell instances to be reported for unconnected ports to be connected to either the scan enable signal or the specified signals. The instance_object can be a single instance pathname, a TCL list of instance pathnames, or a collection of instance objects as returned by the get_instances command.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example connects the unconnected scan enable ports of specified clock gating instances. The first instance is connected to the sen1 pin, which drives the signal with the active state set to low. The next two instances are connected to the default scan enable signal, sen.

The results of these commands are shown by the report_clock_gating command output:

```
set_scan_enable sen
set_clock_gating -instance clkg1/clkg1/clkgLA -driver sen1 -active low
set_clock_gating -instance {clkg3/clkg1/clkgLA clkg2/clkg1/clkgLA}
set_system_mode analysis

// Note: The following clock gating instances have unconnected ports that
will be connected to a scan enable signal.
---------------------------------------------------------------------------
Clock Gating            Unconnected       Signal
Instance                Port              Driver
---------------------------------------------------------------------------
clkg1/clkg1/clkgLA      SE                sen1
clkg2/clkg1/clkgLA      SE                sen
clkg3/clkg1/clkgLA      SE                sen
---------------------------------------------------------------------------
.......
insert_test_logic
report_clock_gating -instance [get_instances clkg* -below_instances
                                                          clkg*/clkg1]

---------------------------------------------------------------------------
Clock Gating            Unconnected       Signal
Instance                Port              Driver
---------------------------------------------------------------------------
clkg1/clkg1/clkgLA      SE                sen1
clkg2/clkg1/clkgLA      SE                sen
clkg3/clkg1/clkgLA      SE                sen
---------------------------------------------------------------------------
```

Related Topics

set_clock_gating

report_clocks

Context: all contexts

Mode: setup, analysis

Usage

report_clocks [-All | -User | -Sdc] [-Display {[DEBug] [DESign] [DAta] [Wave] | ALl} [APpend]] [{> | >>} *file_pathname*]

Description

Displays a list of all user-defined clocks and SDC-defined clocks.

The report_clocks command lists all user-defined clocks created with the add_clocks command or all SDC-defined clocks created by an SDC file loaded with the read_sdc command or both.

Arguments

- -All | -User | -Sdc

  A 1-of-3 choice of optional switches that report user-defined clocks or SDC clocks or both. User-defined clocks are created by the add_clocks command and SDC clocks are created by the SDC file loaded with the read_sdc command. -All is the default option when the flat model is available. Note that the -Sdc switch is valid only when the flat model is available.

  For information about the user-defined clocks report, refer to Example 3. For information about the SDC clocks report, refer to Example 4.

- -Display {[DEBug] [DESign] [DAta] [Wave] | ALl} [APpend]

  An optional switch and repeatable literal that specify to display the instance(s) in the specified window; if Append is specified, the instances are added to the existing contents of the window instead of replacing the contents. The window choices are as follows:

  DEBug
  Specifies the Debug window. This is the default.

  DESign
  Specifies the Design window.

  DAta
  Specifies the Data window.

  Wave
  Specifies the Wave window.

  ALl
  Specifies all supported windows.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of file_pathname.

- *>> file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example 1

The following example adds two clocks to the clock list and then displays a list of the clocks:

add_clocks 1 clk1
add_clocks 0 clk0
report_clocks

```
User-defined Clocks (2):
 ---------  -----------  --------  -------  ----------------
 Off State  Constraints  Internal  Name     Other Properties
 ---------  -----------  --------  -------  ----------------
     1                      No     clk1
     0                      No     clk2
```

Example 2

The following example adds two clocks to the clock list and then displays a list of the clocks:

add_clocks 1 clk_0 clk_1 clk_2 clk_3 -internal -pin_name group_clk

```
// Note: Adding primary input group_clk merging 4 nets.
```

report_clocks

```
User-defined Clocks (1):
 ---------  -----------  --------  ---------   ---------------------  ------
 Off State  Constraints  Internal  Name        Other Properties       Pin(s)
 ---------  -----------  --------  ---------   ---------------------  ------
     1                     Yes    group_clk   (merged internal pin)  clk_0
                                                                      clk_1
                                                                      clk_2
                                                                      clk_3
```

Example 3

The following example reports only user-defined clocks:

> **report_clocks -user**

```
User-defined Clocks (4):
---------  --------------  ------------  --------  ------------- ----------------- ------------
Off State  Constraints     Type          Internal  Name          Other Properties  Pin(s)
---------  --------------  ------------  --------  ------------- ----------------- ------------
   0       Free Running                  Yes       /CG/occ/gclk                    /CG/occ/gclk
   0       Asynchronous                  No        /refclk       (period:1.67ns)
   1       Always Capture  shift-MASTER  No        /clk
   1       -               shift-MASTER  Yes       /Extclk0                        /CG/Extclk0
```

The following explains the columns in the report:

- Off State — Lists the off state of the clock as defined by the add_clocks command.
- Constraints — Lists the special constraint of the clock specified with the add_clocks command:
  - Free Running — The clock is defined with the -free_running switch.

  - Asynchronous — The clock is defined with the -free_running and -period switches.
  - Always Capture — The clock is defined with the -always_capture switch.
- Type — Lists the type of clocks learned by DRC. This is reported only when the tool is in a non-setup mode.
- Internal — Indicates whether the clock is defined on an internal pin (using the “add_clocks -internal” command).
- Name — Lists the name of the clock.
- Other Properties — Lists any additional properties of the clock.
- Pin(s) — Lists one or more internal pins that are connected to the clock.

Example 4

The following example reports only SDC clocks:

> **report_clocks -sdc**

```
SDC-defined Clocks (3):
------------  ---------- ------------ ---------  ------ ----------------------------------
Type          Add_clocks Create_clock Divide_by   Name    Nodes
------------  ---------- ------------ ---------  ------ ----------------------------------
Source             X            X                clk1x  /clk1 (1)
  Generated                                 2     clk2x  /CG/occicg_clk2x_top_inst/gclk (26)
  Generated                                 4     clk4x  /CG/occicg_clk4x_top_inst/gclk (36)
```

The report groups the generated clocks under the source clock from which they’re derived. The following explains the columns in the report:

- Type— Lists the type of SDC clock, which is either a Source clock (from SDC create_clock command) or a Generated clock (from the SDC command create_generated_clock).
- Add_clocks — Indicates if the clock is also defined with the add_clocks command.
- Create_clock — Indicates when a source clock is defined using SDC create_clock command.
- Divide_by — Lists the divide_by ratio of the clock (from the SDC command “create_generated_clock -divide_by”).
- Name — Lists the name of the clock.
- Nodes — Lists the target nodes of the clock.

Example 6

The following example adds two internal clocks and then shows the output of the report clocks report_clocks command.

SETUP> **add_clocks 0 -pin_name clock_1x_int -internal clk1_inst/Y clk2_inst/Y clk3_inst/Y**
SETUP> **add_clocks 0 -pin_name clock_2x_int -internal clk4_inst/Y clk5_inst/Y clk6_inst/Y**
SETUP> **report_clocks**

```
User-defined Clocks (2):
 ---------  -----------  --------  ------------  ---------------------  ------
 Off State  Constraints  Internal  Name          Other Properties       Pin(s)
 ---------  -----------  --------  ------------  ---------------------  ------
    0                   Yes       clock_2x_int (merged internal pin) clk4_inst/Y
                                                                     clk5_inst/Y
                                                                     clk6_inst/Y
    0                   Yes       clock_1x_int (merged internal pin) clk1_inst/Y
                                                                     clk2_inst/Y
                                                                     clk3_inst/Y
```

Related Commands

add_clocks
analyze_control_signals
delete_clocks
read_sdc

report_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_compactor_connections [-All_blocks | {-CHANnel *channel_number*} | {-CHAIn *chain_name*}] [{> | >>} *file_pathname*]

Description

Displays defined scan chain connections for each EDT scan channel output.

If issued from Setup mode, the report_compactor_connections command lists scan chain connections for EDT scan channel outputs previously defined using the set_compactor_connections command.

If issued from analysis mode, all compactor connections are reported, whether defined by Tessent TestKompress or with the set_compactor_connections command.

You can display the chain connection information for all scan channel outputs, or optionally include the -Channel or -Chain switch to display the information only for a particular scan chain or channel.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the chain connection information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays information for all the EDT scan channel outputs in every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the “Modular Compressed ATPG” chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display all chain connection information for all EDT blocks. Without this switch, the tool displays the information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- **-**CHANnel *channel_number*

  An optional switch and positive integer pair that specify to display the scan chain connections for only the scan channel identified by *channel_number*.

- **-**CHAIn *chain_name*

  An optional switch and string pair that specify to display the scan chain connections for only the scan channel whose compactor group includes *chain_name*. Use this switch to see which scan channel is associated with a particular chain.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

Assume a design has eight scan chains named “chain1” through “chain8”, whose outputs currently have user-specified connections through the compactor to two EDT scan channels. These user-defined connections are the result of previous set_compactor_connections commands.

The following example displays the compactor connection information for the chain3 scan chain. You can see from the report that chain3 is part of a compactor group that also includes chain4 and chain5; also that the outputs of these three chains are currently specified to be compacted into the output of scan channel number 2.

report_compactor_connections -chain chain3

```
//    Reporting user-defined compactor connections:
//
//    Channel     Chains
//    -------     ------
//       2        chain3 - chain5
//
```

Example 2

The following example looks up the connection information for scan channel number 1.

report_compactor_connections -channel 1

```
//    Reporting user-defined compactor connections:
//
//    Channel     Chains
//    -------     ------
//       1        chain1, chain2, chain6, chain7, chain8
//
```

Related Commands

reset_compactor_connections

set_compactor_connections

report_config_data

Context: all contexts including unspecified

Mode: all modes

Usage

report_config_data [config_object [-in_wrapper *wrapper_object_or_name*]
[-partition *partition_name*]}] [-show_unspecified]
[-levels integer | all] [-indent *integer*]

Description

Reports the content of configuration wrappers in the transcript as it appears inside the file when the configuration data is written to file using the write_config_data command.

Like all report commands, you can use the > command to redirect the output into a file.

Arguments

- ***config_spec***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When config_spec is a collection, the -in_wrapper and -partition options cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the config_spec value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)” and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- -in_wrapper *wrapper_object_spec*

  An optional option value pair that is used to specify the parent wrapper in which the configuration element exists. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The names in the config_element_name_list are relative to the specified parent wrapper.

- -partition *partition_name*

  An optional option value pair that is used to specify the parent wrapper in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications (DftSpecification and DefaultsSpecification/DftSpecification) are located. You can access the Core wrappers comprising the Tessent Core Description syntax in the tcd partition. Using the partition

names meta: and meta:tcd allows you to access the metadata configuration elements that define the default and tcd partitions. Table 7-23 summarizes the available partition names.

Table 7-23. Available Partition for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the Default partition is access which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tessent Core Description syntax is accessed
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

Example 1

The following example creates some configuration elements and then reports them. The last command uses the > myfile to output the reported data into a file called *myfile*.

```
> add_config_element tmp(1)
{/tmp(1)}
> add_config_element tmp(1)/p1 -value v1 -type property
{/tmp(1)/p1}
> add_config_element tmp(1)/p2 -value v2 -type property
{/tmp(1)/p2}
> add_config_element tmp(1)/w1 -type wrapper
{/tmp(1)/w1}
> add_config_element tmp(1)/w1/P1 -value v1 -type property
{/tmp(1)/w1/P1}
> add_config_element tmp(1)/w1/P2 -value v2 -type property
{/tmp(1)/w1/P2}
> add_config_element tmp(1)/w2 -copy_from tmp(1)/w1
{/tmp(1)/w2}
> report_config_data

tmp(1) {
  p1 : v1;
  p2 : v2;
  w1 {
    P1 : v1;
    P2 : v2;
  }
  w2 {
    P1 : v1;
    P2 : v2;
```

```
    }
  }
  > report_config_data tmp(1) -level 1

  tmp(1) {
    p1 : v1;
    p2 : v2;
    w1 {
      // Not shown
    }
    w2 {
      // Not shown
    }
  }
  > report_config_data tmp(1) -level 1 > myfile
```

Example 2

This example shows how config data is reported when it is of type CSV. The read_config_data is used to parse a CSV file generated from Excel. Notice how the format of the data is changed when only reporting the content of the CSV wrapper as opposed to the CSV Wrapper itself. The same feature exists in the write_config_data command such that you can export CSV data back to Excel.

```
> add_config_element csv(1)
CDV(1)
> read_config_data mydata.csv -in csv(1)
> report_config_data
csv(1) {
  "", "", "", "", "";
  a, b, c;
  "a,c", d;
  "", "", "";
}
> report_config_data csv(1)

,,,,
a,b,c
"a,c",d
,,
```

Related Commands

report_config_data

write_config_data

report_config_messages

Context: all contexts

Mode: all modes

Usage

report_config_message **config_spec**
[-hierarchical]
[-all | [-error] [-warning] [-info]]

Description

Reports messages associated to a configuration element. When the -hierarchical option is used, it also reports the messages associated to elements below the specified wrapper.

Arguments

- *config_object_spec*

  A required value that specifies the name of a configuration element or a collection containing one configuration element.

- -hierarchical

  An optional switch that can only be used when the config_object_spec is a wrapper and specifies that the command should also return elements below the specified wrapper having at least one config message of the specified types.

- -all

  An optional Boolean option that specifies that all message types are to be returned. The use of the -all option is mutually exclusive to the specification of the -error, -warning and -info options. When none of the four options is specified, -all is assumed.

- -error

  An optional Boolean option that specifies that the messages to be returned are of type error. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -warning

  An optional Boolean option that specifies that the messages to be returned are of type warning. The use of the -error option is mutually exclusive to the specification of the -all, -warning, and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

- -info

  An optional Boolean option that specifies that the messages to be returned are of type info. The use of the -error option is mutually exclusive to the specification of the -all, -warning,

and -info options. When none of the -error, -warning and -info options is specified, -all is assumed.

Example

The following example adds configuration messages of type error, warning and info on some elements and reports them.

```
> add_config_element tmp(1)
{/tmp(1)}
> add_config_element tmp(1)/p1 -value v1 -type property
{/tmp(1)/p1}
> add_config_element tmp(1)/p2 -value v2 -type property
{/tmp(1)/p2}
> add_config_element tmp(1)/w1 -type wrapper
{/tmp(1)/w1}
> add_config_element tmp(1)/w1/P1 -value v1 -type property
{/tmp(1)/w1/P1}
> add_config_element tmp(1)/w1/P2 -value v2 -type property
{/tmp(1)/w1/P2}
> add_config_element tmp(1)/w2 -copy_from tmp(1)/w1
{/tmp(1)/w2}
> add_config_message "Message1" -config_object tmp(1)/w1 -error
{/tmp(1)/w1}
> add_config_message "Message2" -config_object tmp(1)/w1/P2 -info
{/tmp(1)/w1/P2}
> add_config_message "Message3" -config_object tmp(1)/w1/P2 -warning
{/tmp(1)/w1/P2}
> report_conf_message tmp(1)/w1

Report for errors, warnings, infos
  /tmp(1)/w1
    E0) Message1

> report_conf_message tmp(1)/w1 -hierarchical

Report for errors, warnings, infos
  /tmp(1)/w1
    E0) Message1
  /tmp(1)/w1/P2
    W0) Message3
    I0) Message2
```

Related Commands

add_config_message

get_config_messages

report_config_syntax

Context: all contexts

Mode: all modes

Usage

report_config_syntax *config_or_meta_object_spec*
[-properties_only | -wrappers_only]
[-levels *integer*]

Description

Reports the available configuration syntax found below the specified config_or_meta_object_spec element.

If you refer to the meta path, you only need to specify the wrapper and property names such as DftSpecification/IjtagNetwork even though the real path name is Wrapper(DftSpecification)/Wrapper(IjtagNetwork). If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it will look up the metadata of the element and report its syntax.

Three optional options limit the amount of syntax displayed to only wrapper, only properties, or to a specified sub-level count.

Arguments

- *config_or_meta_object_spec*

  A required string that specifies the configuration wrapper for which you want to see the available syntax. If you refer to the meta path, you only need to specify the wrapper and property names such as “DftSpecification/IjtagNetwork” even though the real meta path name is “Wrapper(DftSpecification)/Wrapper(IjtagNetwork)”. If you have a configuration element such as DftSpecification(corea,rtl)/IjtagNetwork, you can use it too and it will look up the metadata of the element and report its syntax.

- -properties_only

  An optional Boolean option that restricts the command to only show the properties at the top level of the specified wrapper.

- -wrappers_only

  An optional Boolean option that restricts the command to only show the wrappers and exclude the properties.

- -level *integer*

  An optional option value pair that specifies the number of sub-wrappers to show. When unspecified, all sub-levels are shown.

Example

The following example uses the report_command to show the available syntax currently avalable in the DftSpecification wrapper limiting the levels to 2.

The second example shows the entire syntax available in the Tdr wrapper.

```
> set_context dft -rtl
> report_config_syntax DftSpecification -level 2
DftSpecification(<module_name>,<id>) {
  rtl_extension               : <string>  ; // default: v
  gate_extension              : <string>  ; // default: vg
  reuse_modules_when_possible : <boolean> ; // default: auto
  IjtagNetwork {
    DataInPorts {
    }
    DataOutPorts {
    }
    HostScanInterface(<id>) {
    }
  }
}
> report_config_syntax \
      DftSpecification/IjtagNetwork/HostScanInterface/Tdr
DftSpecification(<module_name>,<id>) {
  IjtagNetwork {
    HostScanInterface(<id>) {
      Tdr(<id>) {
        Interface {
          tck            : <port_name>       ; // default: ijtag_tck
          reset          : <port_name>       ; // default: ijtag_reset
          select         : <port_name>       ; // default: ijtag_sel
          shift_en       : <port_name>       ; // default: ijtag_se
          capture_en     : <port_name>       ; // default: ijtag_ce
          update_en      : <port_name>       ; // default: ijtag_ue
          scan_in        : <port_name>       ; // default: ijtag_si
          scan_out       : <port_name>       ; // default: ijtag_so

          reset_polarity : <active_polarity> ; // legal : active_high
(active_low)
        }
        DataInPorts {
          count       : <int>            ; // default: auto
          port_naming : <port_name>, ... ; // default: ijtag_data_in[%d]
          connection(<range>) : <pin_name> ; // repeatable
        }
        DataOutPorts {
          count        : <int>            ; // default: auto
         port_naming  : <port_name>, ... ; // default: ijtag_data_out[%d]
          multiplexing : <boolean>        ; // default: auto
          output_timing(<range>) : <timing_spec> ; // repeatable
          connection(<range>) : <pin_name> ; // repeatable
        }
        parent_instance              : <instance_name>      ;
        leaf_instance_name           : <leaf_instance_name> ;
       keep_active_during_scan_test : <boolean>        ; // default: auto
       length                      : <int>              ; // default: auto
       extra_bits_capture_value     : <enum>       ; // legal : 1 (0) self
```

```
            reset_value                  : <binary>     ; // default: auto
            Attributes {
              <attribute> : <string> ; // repeatable
            }
            DecodedSignal(<signal_name>) {
              decode_values : <binary_code>, ... ;
              multiplexing  : <boolean>          ; // default: auto
             output_timing : <timing_spec>; // legal : (auto) unlatched normal
    ijtag_scan_selection
              connection : <port_pin_name> ; // repeatable
              Attributes {
                <attribute> : <string> ; // repeatable
              }
            }
          }
        }
      }
    }
```

Related Commands

report_config_data

report_context

Context: unspecified, all contexts

Mode: all modes

Usage

report_context

Description

Reports the current context as specified by the set_context command and any inferred subcontexts.

This command also provides information about inferred sub-contexts as shown in the examples below.

Arguments

- None

Return Value

None

Example 1

In the following example, the command reports that it ran in uncompressed ATPG mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Uncompressed ATPG (no EDT) mode active.
```

Example 2

In the following example, the command reports that it is in extraction mode.

report_context

```
// Context: patterns -ijtag -no_rtl
// ICL extraction mode active.
```

Example 3

In the following example, the command reports that it is in compressed ATPG mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Compressed ATPG (EDT) mode active.
```

Example 4

In the following example, the command reports that it is in Logic BIST mode.

report_context

```
//  Context: patterns -scan -no_rtl
//  Logic BIST mode active.
```

Related Commands

get_context

set_context

report_control_signals

Context: dft -scan

Mode: analysis

Usage

report_control_signals {[-All] | {[*pin_pathname*…] [-Clock] [-Set] [-Reset] [-Write] [-Read] [-Tristate_enable]} [-Verbose] [-NOSTABLE_High] [-NOSTABLE_Low]

Description

Displays the rules checking results for control signals consisting of clocks added with the add_clocks command and pins identified for gating scannable memory elements with test logic.

Arguments

- -All

  An optional switch that outputs all control signals to the report. Default setting.

- *pin_pathname*

  An optional, repeatable string that specifies the pathnames of control signals to report on.

- -Clock

  An optional switch that outputs clock control signals in the report.

- -Set

  An optional switch that reports on all set control signals.

- -Reset

  An optional switch that reports on all reset control signals.

- -Write

  An optional switch that reports on all write control signals.

- -Read

  An optional switch that reports on all read control signals.

- -Tristate_enable

  An optional switch that reports on all tristate enable signals.

- -Verbose

  An optional switch that includes all memory elements associated with each control signal in the report.

- -NOSTABLE_High

  An optional switch that disables the report on any stable-high memory elements.

- -NOSTABLE_Low

  An optional switch that disables the report on any stable-low memory elements.

Related Commands

add_clocks
delete_clocks
report_dft_check

report_core_descriptions

Context: all contexts

Mode: all modes

Usage

report_core_descriptions

Description

Reports all core description files.

The tool uses the information in this file to retarget patterns for instances of the associated core.

Arguments

None.

Example

The following example shows the output of the report_core_descriptions command in two modes:

In setup mode:

```
report_core_descriptions

Name           Mode Name     Mode Type
------------   -----------   -----------
* chip         retargeting   retargeting

Note: * denotes current design
```

In analysis mode:

```
report_core_descriptions

Name               Mode        Mode        Instantiations  Chain     Scan
                   Name        Type                        Patterns  Patterns
----------------- ----------- ----------- --------------  --------  --------
* chip             retargeting retargeting
  piccpu_maxlen16 internal    internal    1               8         117

Note: * denotes current design
```

Related Commands

add_core_instances
delete_core_instances
report_core_instances
write_core_description

report_core_instances

Context: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Usage

report_core_instances [**-cores *core_names*** | -modules *module_objects* |
-instances *instance_objects*] [-silent]

Description

Reports the instance bindings created with the add_core_instances command, as well as the pattern count for patterns read in for each core.

If this command is invoked in analysis mode, the pattern count for each core is also reported.

You can specify the absolute instance path names or the module or core names. You can either report all core instances that have been added (the default), or filter by modules, instances, or core names.

Arguments

- **-cores *core_names***

  A switch and Tcl list that specifies the core names of the core descriptions from which all instance bindings should be reported.

- -modules *module_objects*

  An optional switch and string pair that specifies an object name, a Tcl list of object names (hierarchical names), or a collection of objects. This command reports the instance bindings for each instantiation of those modules in the design.

- -instances *instance_objects*

  An optional switch and string pair that specifies an object name, a Tcl list of object names (hierarchical names), or a collection of objects. This command reports the instance bindings for each instance in the list.

- -silent

  An optional switch that specifies to suppress warning messages.

Example 1

The following example reports all existing core instances while in setup mode.

report_core_instances

```
Instance  Core       Mode Name   Mode Type   Module
--------  ---------  ---------   ---------   -----------
core_1    my_core_1  internal    internal    my_module_1
core_2    my_core_2  internal    internal    my_module_2
core_3    my_core_3  internal    internal    my_module_3
```

Example 2

The following example reports all existing core instances while in analysis mode.

report_core_instances

```
Instance Core       Mode Name  Mode Type  Module      Chain      Scan
                                                      Patterns   Patterns
-------- ---------  ---------  ---------- ----------- --------   --------
core_1   my_core_1  internal   internal   my_module_1 12         10
core_2   my_core_2  internal   internal   my_module_2 12         10
core_3   my_core_3  internal   internal   my_module_3 6          74
```

Example 3

The following example first reports all core instances for modules my_module_1 and my_module_2 while in setup mode; it then reports the same while in analysis mode.

report_core_instances -modules {my_module_1 my_module_2}

```
Instance Core       Mode Name  Mode Type  Module
-------- ---------  ---------  ---------- -------------
core_1   my_core_1  internal   internal   my_module_1
core_2   my_core_2  internal   internal   my_module_2
```

set_system_mode analysis
...
report_core_instances -modules {my_module_1 my_module_2}

```
Instance Core       Mode Name  Mode Type  Module      Chain      Scan
                                                      Patterns   Patterns
-------- ---------  ---------  ---------  ----------- --------   --------
core_1   my_core_1  internal   internal   my_module_1 12         10
core_2   my_core_2  internal   internal   my_module_2 12         10
```

Example 4

The following example first reports core instances core_1 and core_3 while in setup mode; it then reports the same while in analysis mode.

report_core_instances -instances {core_1 core_3}

```
Instance Core       Mode Name  Mode Type   Module
-------- ---------- ---------  ---------- -----------
core_1   my_core_1  internal   internal   my_module_1
-------- ---------- ---------  ---------- -----------
core_3   my_core_3  internal   internal   my_module_3
```

set_system_mode analysis
...
report_core_instances -instances {core_1 core_3}

```
Instance Core      Mode Name  Mode Type  Module       Chain      Scan
                                                      Patterns   Patterns
-------- --------- ---------  ---------  ------       --------   --------
core_1   my_core_1 internal   internal   my_module_1  12         10
-------- --------- ---------  ---------  ------       --------   --------
core_3   my_core_3 internal   internal   my_module_3  6          74
```

Related Commands

add_core_instances
delete_core_instances
read_core_descriptions
write_core_description

report_design_sources

Context: all contexts

Mode: all modes

Usage

report_design_sources

Description

Reports the pathnames or file extensions previously specified with the set_design_sources command.

Arguments

None

Return Value

None

Example

The following example shows a report generated by this command:

set_design_sources -format icl -y ../data/icl_primitives -extension icl

set_design_sources -y ../data/verilog_primitives -extension v

report_design_sources

```
//  ICL search_design_load_path: activated
//  ---------------------------------------------------------------
//  format   type  path                          file extensions
//  -------  ----  ----------------------------  ---------------
//  Verilog  dir   '../data/verilog_primitives'  'v v.gz'
//  ICL      dir   '../data/icl_primitives'      'icl icl.gz'
```

Related Commands

get_design_sources

set_design_sources

report_dfm_rules

Context: patterns -scan_diagnosis

Mode: analysis

Usage

report_dfm_rules [-header | -col *columns*...]

Description

Returns a tabular report of the DFM rules contained in the layout database.

By default, all columns are listed. These are: id, rule, layer, and type.

For more information about DFM rules, see "Diagnosis for Design for Manufacturability Analysis."

Arguments

- -header

  An optional switch that returns a list of available column names.

- -col *columns*...

  An optional switch that specifies the column names you want to use for the DFM rule information in the report.

Example1

The following example creates a report that contains all available columns.

report_dfm_rules

```
-----------------------------------------------------------
  id   rule                                layer     type
-----------------------------------------------------------
  1    minimum_space_4_line_route1         route_1   BRIDGE
  2    metal_cross_edge_route1             route_1   OPEN
  3    low_density_min_space_route2        route_2   BRIDGE
  4    half_comb_route3                    route_2   OPEN
  5    metal_cross_edge_route1             route_3   OPEN
  6    corner_to_interior_metal4           route_4   BRIDGE
  7    divergent_line_with_jog_metal5      route_5   BRIDGE
  8    metal_cross_edge_route1             route_5   OPEN
  9    route_5_barbell                     route_5   OPEN
  10   route_2_between_vias                route_2   OPEN
-----------------------------------------------------------
```

Example 2

The following example creates a report that contains the rule name and the type.

report_dfm_rules -col rule type

```
-------------------------------------------
  rule                                type
-------------------------------------------
  minimum_space_4_line_route1       BRIDGE
  metal_cross_edge_route1           OPEN
  low_density_min_space_route2      BRIDGE
  half_comb_route3                  OPEN
  metal_cross_edge_route1           OPEN
  corner_to_interior_metal4         BRIDGE
  divergent_line_with_jog_metal5    BRIDGE
  metal_cross_edge_route1           OPEN
  route_5_barbell                   OPEN
  route_2_between_vias              OPEN
-------------------------------------------
```

Related Commands

get_dfm_rules

report_dft_check

Context: dft -scan

Mode: analysis

Usage

report_dft_check [-All | *instance_pathname*] [-FUll | -Scannable | -Nonscannable | {-Defined {Scan | Nonscan} | -Identified | -Unidentified | {-RUle {S1 | S2 | S3 | S4}} | -Tristate | -RAm] [{> | >>} *file_pathname*]

Description

Generates scannability check information for all non-scan instances in the design, or within a specific hierarchical instance.

The displayed or written report includes six columns of information as described here:

- The first column displays whether the DFT Rules Checker identified the non-scan instance as scannable or non-scannable.
- The second column displays whether the non-scan instance is unidentified, identified by the scan identification process, or defined as non-scannable or scannable with the add_nonscan_instances and add_scan_instances commands, respectively.
- The third column displays the clock that is associated with the non-scan instance, or displays “Test-Logic” to specify that test logic will be added to make the instance scannable. If the non-scan instance is identified as non-scannable, the third column displays the design rule ID number, where the non-scan instance failed the clock rules. The gate index number of the non-scannable instance is also shown, as well as the set, reset or clock primary inputs to the memory element that failed the rules checking.
- The fourth column displays the instance name of the non-scan instance.
- The fifth column displays the library model name associated with the instance.
- The sixth column displays “Stable-high” if the clock inputs of the non-scan instances are at a one-state, with respect to the clock primary inputs.

An example of the output of this command, along with additional information, is covered in “Reporting Scannability Information” in the *Tessent Scan and ATPG User’s Manual*.

Arguments

- -All

  An optional switch that specifies to display all non-scan instances for the entire design. This is the default.

- *instance_pathname*

  An optional string that specifies to report scannability information for the specified instance. If the instance pathname specifies a hierarchical instance, the tool generates scannability

information for all instances within that hierarchical block. If the instance pathname specifies a particular sequential instance, the tool generates scannability information for only that instance.

- -Full

  An optional switch that specifies to display the scannability check information for all non-scan instances. This is the default.

- -Scannable

  An optional switch that specifies to display the non-scan instances that the tool has identified during the rules checking process to be scannable.

- -Nonscannable

  An optional switch that specifies to display the non-scan instances that the tool has identified during the rules checking process to be non-scannable.

- -Defined Scan | Nonscan

  An optional switch and literal pair that specifies whether to display user-defined scan or non-scan instances. The valid literals are as follows:

  Scan — A literal that specifies to display scan instances defined with the add_scan_instances command.

  Nonscan — A literal that specifies to display non-scan instances defined with the add_nonscan_instances command.

- -Identified

  An optional switch that displays identified non-scan instances.

- -Unidentified

  An optional switch that displays unidentified non-scan instances.

- -RUle S1 | S2 | S3 | S4

  An optional switch and literal that specifies which non-scannable cell violations to report. For more information on S1, S2, S3, and S4 rule violations, refer to "Scannability Rules (S Rules)."

- -Tristate

  An optional switch that displays the enable lines of tri-state gates connected to the outputs of memory elements.

- -RAm

  An optional switch that displays the RAM gates identified to be controllable through test logic insertion.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list that for creates *or* replaces the contents of file_pathname.

- *>> file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append to the contents of file_pathname.

Examples

The following example displays the scannability check for all non-scan instances in the design:

add_clocks 1 clk1
add_clocks 0 clk0
set_system_mode analysis
report_dft_check

```
SCANNABLE     DEFINED-NONSCAN  /CLK1 /U1   FD1
SCANNABLE     IDENTIFIED       /CLK1 /U2   FD1
SCANNABLE     UNIDENTIFIED     /CLK1 /U3   FD1
SCANNABLE     DEFINED-SCAN     /CLK1 /U4   FD1
SCANNABLE     UNIDENTIFIED     /CLK2 /U5   FD1
SCANNABLE     IDENTIFIED       /CLK2 /U6   FD1
NON-SCANNABLE UNIDENTIFIED      S1   /U7   FD2 (34)
   Clock #1:   /CLK3 (11)
Number of non-scannable instances fails on S1 rule = 1
Number of instances found = 1
Number of instances reported = 1
```

Related Commands

echo
report_drc_rules
report_seq_transparent_procedures

report_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Usage

report_diagnosis [-ENCoded] [-SHORT]

Description

Writes the scan diagnosis report to standard I/O.

Arguments

- -ENCoded

  An optional switch encoding the suspect's pin pathname, net name, and cell name in the diagnosis report.

- -SHORT

  An optional switch that suppresses the display of layout-aware diagnosis information in the diagnosis report.

Related Commands

write_diagnosis

report_display_instances

Context: all contexts

Mode: setup, analysis

Usage

report_display_instances {***gate_id#*** | ***instance_name***}… | **-All**
[-Display {DEBug | DESign | DAta}] [-Full]

Description

Shows netlist information for the specified instances displayed in the Debug, Design, or Data window.

The report_display_instances command reports, in the session transcript, netlist information for the specified instances currently displayed in the DFTVisualizer Debug, Design, or Data window. The default report includes the information listed in the following table:

Table 7-24. Default Netlist Information Reported for Instances

"set_gate_level Design" in effect	**"set_gate_level Primitive" in effect**
Instance name	Instance name
DFT library model name	Gate identification number
	Gate type

You can display the same information from a tool's command line using the report_gates command, whether or not DFTVisualizer is open.

Arguments

- ***gate_id#***

  A repeatable integer that specifies a gate whose netlist information you are reporting. The value of the gate_id# argument is the unique identification number the tool automatically assigns to every gate within the design during the model flattening process.

- ***instance_name***

  A repeatable string that specifies the name of an instance whose netlist information you are reporting.

- **-All**

  A switch that specifies to report all instances in the current display netlist (including the compacted gates).

- -Display {DEBug | DESign | Data}

  A switch and literal that specify the window containing the instance(s) you are reporting. The window choices are as follows:

DEBug
Specifies the Debug window. This is the invocation default.

DESign
Specifies the Design window.

Data
Specifies the Data window.

- -Full

  A switch that specifies to include the following information for each pin on each reported instance:

  - Pin name
  - Pin type (input or output)
  - Simulated pin data (if appropriate)
  - Gates to which that pin connects

Examples

The following example analyzes a DRC violation (automatically invoking DFTVisualizer in the process and displaying the part of the netlist related to the violation in the Debug window):

set_gate_level design
analyze_drc_violation c3-1

```
//  command: open_visualizer -display debug
//  Note: Gate report now set to clock_cone (clock=/tck).
//  Creating Schematic for 5 instances
```

The DFTVisualizer Transcript window shows the following information for the violation:

```
=============== (Violation ID = c3-1) ================
//  Note: Clock /tck failed rule C3 on input 3 of /reg_enable (76). (C3-1)
//  Note: Source of violation: input 3 of /reg_pstate_2_ (70).
//  Note: Gate report now set to clock_cone (clock=/tck).
Creating Schematic for 5 instances
```

The next example shows the differences between the default and the detailed report for the gate where clock /tck failed the C3 DRC:

report_display_instances /reg_enable

```
// /reg_enable dffr
```

report_display_instances /reg_enable -full

```
// /reg_enable  dffr
//      D      I  (E)  /ix130/Y
//      CLK    I  (C)  /ix473/Y
//      R      I  (-)  /ix475/Y
//      Q      O  (-)  /tri_pin/E
//      QB     O  (-)
```

Related Commands

add_display_instances
analyze_drc_violation
open_visualizer
report_gates
set_gate_report

report_drc_rules

Context: all contexts

Mode: all modes

Usage

report_drc_rules [-Fails_summary | -Summary |
rule_id… | *rule_id-occurence#*… | -All_fails]
[{> | >>} *file_pathname*]

C1 Usage:

report_drc_rules **C1** [-EXcluded]

D5 Usage:

report_drc_rules **D5**
[{ [-TYpe {I0 | I1 | IX | T0 | T1 | TX | TLA}…]
[-NOType {I0 | I1 | IX | T0 | T1 | TX | TLA}…]
[-EDge_triggered | -LEvel_sensitive] } | -Summary]
[{> | >>} *file_pathname*]

E5 Usage:

report_drc_rules **E5** [-TIMing_exceptions]

Description

Displays either a summary of DRC violations (fails) or violation occurrence message(s).

This command can display a report in one of two formats:

- Summary report—Lists for each reported design rule, one line of data per rule, the current number of DRC violations (fails), the violation handling, and a brief description of the rule. When you request a summary of D5 violations, the report lists the number of D5 violations of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X, TLA).
- Occurrence report—Lists one or more violation occurrence messages that give details of specific DRC violations

Table 7-25 provides a summary of the available information displayed and the arguments you use to obtain them. Refer to the Arguments subsection for complete details about the arguments.

Table 7-25. Available Information Displayed and Arguments

Desired Display	Rules/Occurrences Covered	Argument
Summary report	Design rules that resulted in violations (fails) during DRC	-Fails_summary
	All design rules	-Summary

Table 7-25. Available Information Displayed and Arguments

Desired Display	Rules/Occurrences Covered	Argument
Occurrence report	Specific rule, specific occurrence	*rule_id-occurence#*
	Specific rule, all occurrences[1]	*rule_id*
	All occurrences	-All_fails

1. Additional D5-specific arguments enable you to further customize the D5 information display.

You can use the set_drc_handling command to change the handling of many of the A (RAM), C (clock), D (data), E (extra), K (EDT), T (trace), and W (timing) rules.

For more information about the design rules, refer to "Design Rule Checking" on page 2135.

Arguments

- -Fails_Summary

  A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during DRC:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule
  - Brief description of the rule

  This is the default.

Note

This switch does not display anything if there are no rule violations or the tool has not yet performed DRC.

- -Summary

  A switch that specifies to display a summary report that varies depending on whether you are reporting on all design rules or just on the D5 rule. (The latter reporting is available only in Tessent FastScan and Tessent TestKompress.)

  When you report on all rules (report_drc_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during DRC:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule
  - Brief description of the rule

When you report on just the D5 rule (report_drc_rules D5 -Summary) in Tessent FastScan or Tessent TestKompress, the tool displays the number of non-scan memory elements of each type (INIT-0, INIT-1, INIT-X, TIE-0, TIE-1, TIE-X or TLA) that resulted in a D5 violation during DRC. Refer to the description of the -Type switch for the meaning of the types. See also the "D5" section for detailed information about the D5 rule.

- *rule_id*

  A repeatable string that specifies the identification literal (ID) of a particular design rule for which you want to display the violations. For information about the design rules and their ID, refer to the appropriate section:

  - RAM Rules (A Rules)
  - BIST Rules (B Rules)
  - Clock Rules (C Rules)
  - Scan Cell Data Rules (D Rules)
  - Extra Rules (E Rules)
  - EDT Finder Rules (F Rules)
  - Flattening Rules (FN, FP, and FG Rules)
  - General Rules (G Rules)
  - EDT Rules (K Rules)
  - Procedure Rules (P Rules)
  - Scannability Rules (S Rules)
  - Scan Chain Trace Rules (T Rules)
  - Power-Aware Rules (V Rules)
  - Timing Rules (W Rules)
  - ICL Extraction Rules (I Rules)
  - ICL Semantic Rules (Semantic Rules)
  - Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)

- *rule_id-occurrence#*

  A repeatable string that specifies the identification literal (ID) of a particular design rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. For example, you can analyze the second violation occurrence of the C3 rule by specifying C3-2. Numbers to occurrences of rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.

- -All_Fails

  A switch that specifies to display all occurrence messages for all occurrences of rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a “report_drc_rules <*rule_id*>” command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

C1-Only Arguments

- **C1**

  A required literal that specifies reporting C1 DRC rule violations.

- -EXcluded

  An optional switch for use only with a C1 violation. Specifying this switch reports the C1 violations that have been excluded from the default C1 list because these C1 violations can be handled by the tool without causing potential mismatch.

D5-Only Arguments

- -TYpe I0 | I1 | IX | T0 | T1 | TX | TLA

  An optional switch and repeatable literal that displays D5 occurrence messages for only the specified type(s) of non-scan sequential elements. The literal choices for the type of element are as follows (the term you will see in occurrence messages for each type is shown in parentheses):

  I0 — If the element is at 0 at the beginning of the first capture cycle and may go to any state during capture. (INIT-0)

  I1 — If the element is at 1 at the beginning of the first capture cycle and may go to any state during capture. (INIT-1)

  IX — If the element’s state is unknown at the beginning of the first capture cycle and may go to any state during capture. (INIT-X)

  T0 — If the element is always at 0 during capture. (TIE-0)

  T1 — If the element is always at 1 during capture. (TIE-1)

  TX — If the element is always at an unknown state during capture. (TIE-X)

  TLA — If the element is always transparent when its clock is at its off state. (TLA)

> **Tip**: Except for TLAs, you can also direct the tool to display information for only those D5 elements that are edge-triggered or level-sensitive. See the -Edge_triggered and -Level_sensitive switch descriptions for details.

- -NOType I0 | I1 | IX | T0 | T1 | TX | TLA

  An optional switch and repeatable literal that specify not to display occurrence messages for the particular type(s) of D5 violations. See the description of the -Type switch for the meaning of the literal choices.

- -EDge_triggered | -LEvel_sensitive

  Optional switches that specify to display D5 occurrence messages either for edge-triggered or level-sensitive elements only. The default (when neither option is specified) is to display information for both edge-triggered and level-sensitive elements.

E5-Only Arguments

- -TIMing_exceptions

An optional switch that specifies to report only on E5 violations due to timing exceptions. In order to use this switch you must first issue a "set_drc_handling -timing_exception_check ON" command in setup mode (see "Example 8" on page 1278).

Example 1

The following example displays a summary of the rules that resulted in violations during an attempted ATPG run:

report_drc_rules

```
C3: #fails=2 handling=warning (clock may capture data affected by its
   captured data)
C5: #fails=1 handling=error (clock is connected to multiple ports of same
   latch)
C7: #fails=19 handling=warning (scan cell capture ability check)
C8: #fails=2 handling=warning (PO connected to a clock line)
C9: #fails=2 handling=warning (PO connected to a clock line gated by scan
   cell that uses same clock)
D5: #fails=23 handling=warning (non-scan memory element)
D6: #fails=22 handling=warning (non-transparent non-scan latches)
D7: #fails=22 handling=warning (stable high edge-triggered clock ports)
```

Example 2

The following example displays the occurrence message for the two C3 violations, and the first C7, and first D7 violation reported in the preceding example:

report_drc_rules c3 c7-1 d7-1

```
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_b_i0.latch
   (1501). (C3-1)
// Source of violation: input 3 of /sp_c_i1.slave (1482).
// Warning: Clock /sgst failed rule C3 on input 3 of /sp_c_i0.latch
   (1527). (C3-2)
```

```
// Source of violation: input 3 of /sp_c_i1.slave (1482).
// Warning: Clock input 5 of /c8.master (1495) cannot capture data
   with single clock on. (C7-1)
// Warning: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```

Example 3

The following example changes the handling of data rule 7 (D7) from warning to error and also specifies execution of a full test generation analysis when performing the rules checking for the clock (C) rules:

set_drc_handling d7 error atpg_analysis
set_system_mode analysis

```
//-------------------------------------------------------------
//Begin scan chain identification process, memory elements=8.
//-------------------------------------------------------------
// Reading group test procedure file /user/design/tpf.
// Simulating load/unload procedure in g1 test procedure file.
// Chain = c1 successfully traced with scan_cells = 8.
// Error: Flipflop /FF1 (103) has clock port set to stable high.(D7-1)
// Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

Example 4

The following example displays the occurrence message for the D7 violation:

report_drc_rules d7-1

```
//Error: Flipflop /FF1 (103) has clock port set to stable high. (D7-1)
```

Example 5

The following example displays a summary of the non-scan memory elements that resulted in D5 rule violations:

report_drc_rules d5 -summary

```
//  ---------------------------------------------------------------------
//  44 non-scan memory elements are identified.
//  ---------------------------------------------------------------------
//    9 non-scan memory elements are identified as TIE-0.  (D5)
//    2 non-scan memory elements are identified as TIE-1.  (D5)
//    5 non-scan memory elements are identified as TIE-X.  (D5)
//    3 non-scan memory elements are identified as INIT-0. (D5)
//   11 non-scan memory elements are identified as INIT-1. (D5)
//    4 non-scan memory elements are identified as INIT-X. (D5)
//   10 non-scan memory elements are identified as TLA.    (D5)
//  ---------------------------------------------------------------------
```

Example 6

The following example displays details about the preceding example's D5 violations that occurred on non-scan memory elements the tool identified as type INIT-X:

report_drc_rules d5 -type ix

```
//  Warning: /U1 (78) is a non-scan flip-flop identified as INIT-x.(D5-1)
//  Warning: /U6 (56) is a non-scan flip-flop identified as INIT-x.(D5-7)
//  Warning: /U7 (82) is a non-scan flip-flop identified as INIT-x.(D5-8)
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 4.
```

Example 7

The following example displays the information for just the INIT-X non-scan latch:

report_drc_rules d5 -type ix -level_sensitive

```
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 1.
```

You could obtain the same information using the -Notype switch:

report_drc_rules d5 -notype i1 i0 tx t1 t0 tla -level_sensitive

```
//  Warning: /U9 (11) is a non-scan latch identified as INIT-x.(D5-9)
//  Number of non-scan memory elements reported = 1.
```

Example 8

The following example reads timing exception paths and enables the timing exception check for E5. Next, the example enters analysis mode and reports E5 due to timing exceptions.

SETUP> **read_sdc my_design.sdc**
SETUP> **set_drc_handling E5 -timing_exception_check on**
SETUP> **set_system_mode analysis**
...
// 9 gates may have an observable X-state, 2 of which are due to timing exception paths. (E5)

ANALYSIS> **report_drc_rules e5 -timing_exceptions**
// DFF gate /M3S004FP/VALIDDEVCHIRPK_REG (59193) may have an observable X-state due to timing exception paths. (E5-8)
// DFF gate /M3S014FP0/TXVALIDH_REG (59220) may have an observable X-state due to timing exception paths. (E5-9)

Related Commands

set_drc_handling

report_edt_blocks

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_edt_blocks

Description

Displays a list of the current user-defined EDT block names (also referred to as block tags) and indicates which one, if any, represents the current block.

When the display shows “current block” by a name, it indicates relevant tool commands apply only to the EDT block associated with that name.

You can transfer this restriction (also referred to as the EDT context) to another EDT block with the set_current_edt_block or add_edt_blocks commands.

Example

The following example lists currently defined EDT block names:

report_edt_blocks

```
//  my_core1_block
//  my_core2_block
//  my_core3_block
//  my_core4_block   (current block)
```

Related Commands

add_edt_blocks
delete_edt_blocks
set_current_edt_block

report_edt_configurations

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_edt_configurations [-ALl_blocks] [{> | >>} *file_pathname*]

Description

Displays configuration information for the current compression logic.

Among the items reported are the version, compactor type, power controller status, number of scan channels and related information, multiple compression configurations, and whether the tool is set up to insert bypass logic and lockup cells. This command only displays a partial report from Setup mode. To produce a complete report, you must exit Setup mode to run DRC and flatten the model before issuing this command.

Note

The report_edt_configurations command does not list the names of EDT pins. To display the names of EDT pins, use the report_edt_pins command.

The -all_blocks switch is useful when the netlist read in includes multiple EDT blocks. In this case, the command by default reports the EDT configuration information as described above, but only for the EDT block you have specified as the current EDT block. If you include the -all_blocks switch, the command displays configuration information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the Modular Compressed ATPG chapter of the *Tessent TestKompress User’s Manual*.

Arguments

- -ALl_blocks

  An optional switch that displays EDT configuration information for all EDT blocks. Without this switch, only information for the current EDT block displays. If a current EDT block is not defined, configuration information for all EDT blocks displays.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example 1

The following example reports configuration details for a non-modular design with eight scan chains and one scan channel; first while in Setup mode, then again after DRC:

report_edt_configurations

```
// IP version:                  1
// External scan channels:      1
// Bypass logic:                On
// Lockup cells:                On
// Clocking:                    edge-sensitive
```

set_sytem_mode analysis

```
...
# -------------------------------------------------------
# Begin EDT rules checking.
# -------------------------------------------------------
# Running EDT logic creation Phase.
# 5 pin(s) will be added to the EDT wrapper. (K11)
# EDT rules checking completed, CPU time=0.01 sec.
# All scan input pins were forced to TIE-X.
# All scan output pins were masked.
# ------------------------------------------------------------
...
```

report_edt_configurations

```
// IP version:                  1
// Shift cycles:                381, 373 (internal scan length)
                                + 8 (additional cycles)
// External scan channels:      1
// Internal scan chains:        8
// Masking bits:                2
// Longest chain range:         2 - 373
// Decompressor size:           32
// Shift power controller:      Enabled and inactive(shift power control
                                is off)
// Min switching threshold:     25%
// Compactor type:              Xpress
// Scan cells:                  18650
// Compression per pattern:     16.01x (ATPG bypass = 3 x 6341, EDT = 3 x
                                396)
// Clocking:                    edge-sensitive
// Compactor pipelining:        Off
```

Example 2

The following example defines two EDT blocks in the patterns -scan context, then reports configuration details first for the current block and then for all blocks:

```
# Define the block name "my_block1" for an EDT block
#   and automatically set it as the current EDT block.
```
add_edt_blocks my_block1

```
# Define the location (instance name) of the top-level EDT logic for this
#   block; also the decompressor and compactor.
```

set_edt_instances -edt_logic_top core1_edt_i
set_edt_instances -decompressor core1_edt_decompressor_i
set_edt_instances -compactor core1_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 3

```
# Specify the top-level control signals and channel inputs and outputs for
#   this block (required).
```
set_edt_pins...

```
# Repeat the preceding procedure for another block.
```
add_edt_block my_block2

set_edt_instances -edt_logic_top core2_edt_i
set_edt_instances -decompressor core2_edt_decompressor_i
set_edt_instances -compactor core2_edt_compactor_i

add_scan_chains...
add_scan_chains...

set_edt_options -channels 1

set_edt_pins...

report_edt_configurations

```
//  EDT block "my_block2"
//  --------------------
//  IP version:                     3
//  External scan channels:         1
//  Decompressor size:              10
//  Clocking:                       edge-sensitive
//
//  Note: Only current EDT block "my_block2" is reported. Use "report edt
//  configuration -all_blocks" to report the configurations of all blocks.
```

report_edt_configurations -all_blocks

```
//  EDT block "my_block1"
//  --------------------
//  IP version:                   3
//  External scan channels:       3
//  Decompressor size:            12
//  Clocking:                     edge-sensitive
//
//  EDT block "my_block2"
//  --------------------
//  IP version:                   3
//  External scan channels:       1
//  Decompressor size:            10
//  Clocking:                     edge-sensitive
```

Example 3

The following example reports all EDT blocks. Channel sharing information is shown in red font. Note that "// Blocks sharing input channels:" means the blocks having at least one input channel that is shared with this channel.

report_edt_configurations -all

```
//  EDT block "B1"
//  --------------
//  IP version:                    4
//  Shift cycles:            53,  19 (internal scan length) + 11
// (additional cycles) + 23 (pad cycles due to blocks with longer chains)
//  External scan channels:        2 (1 control input channel)
//  Internal scan chains:          32
//  Masking bits:                  25
//  Longest chain range:           2 - 19
//  Decompressor size:             16
//  Compactor type:                Xpress
//  Blocks sharing input channels: B2
//  Scan cells:                    573
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
//

//  EDT block "B2"
//  --------------
//  IP version:                    4
//  Shift cycles:             53,  2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
//  External input channels:       3 (1 control input channel)
//  External output channels:      2
//  Internal scan chains:          4
//  Masking bits:                  5
//  Longest chain range:           2 - 2
//  Decompressor size:             8
//  Compactor type:                Xpress
//  Blocks sharing input channels: B1, B3
//  Scan cells:                    8
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
//

//  EDT block "B3"
//  --------------
//  IP version:                    4
//  Shift cycles:             53,  2 (internal scan length) + 13
// (additional cycles) + 38 (pad cycles due to blocks with longer chains)
//  External scan channels:        2 (1 control input channel)
//  Internal scan chains:          4
//  Masking bits:                  5
//  Longest chain range:           2 - 2
//  Decompressor size:             8
//  Compactor type:                Xpress
//  Blocks sharing input channels: B2
//  Scan cells:                    8
//  Clocking:                      edge-sensitive
//  Compactor pipelining:          0 stages
```

Related Commands

add_edt_configurations
report_scan_volume
report_edt_pins
delete_edt_configurations
set_edt_options
set_edt_pins

report_edt_connections

Context: dft -edt

Mode: setup, analysis

Usage

report_edt_connections [-All_blocks] [{> | >>} *file_pathname*]

Description

Reports pin connections made during the top-level integration session.

For more information, see "Creating the Top-level Netlist" in the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

  Optional switch that displays pin connections made with the add_edt_connections command in a modular integration session. By default, information for the current EDT block is reported.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example 1

The following example reports pin connections for the current EDT block.

report_edt_connections

```
//
// ---------------------------------------------------------------------
// EDT Signal Connections
// ---------------------------------------------------------------------
// Block Pin description Pin name Connection name
// ----- --------------- -------- ---------------
// B2 Clock edt_clock TK_clk
// B2 Update edt_update TK_upd
// B2 Bypass mode edt_bypass tap_i/tap_ctrl_i/edt_bypass_inst
// B2 Scan channel 1 input edt_channels_in1 TK_channel_in[1]
9
// B2 " " " output edt_channels_out1 TK_channel_out[1]
//
// Note: Only current EDT block "B2" is reported. Use
// "report_edt_connections -all_blocks" to report the connections for all
// blocks.
//
```

```
// ------------------------------------------------------------------
// Generic Connections
// ------------------------------------------------------------------
// Source Destination
// ------ -----------
// tap_i/tap_ctrl_i/test_en B1_inst/test_en
// tap_i/tap_ctrl_i/test_en B2_inst/test_en
//
```

Example 2

The following example reports pin connections for all EDT blocks.

report_edt_connections -all_blocks

```
//
// ------------------------------------------------------------------
// EDT Signal Connections
// ------------------------------------------------------------------
// Block Pin description Pin name Connection name
// ----- --------------- -------- ---------------
// B1 Clock edt_clock TK_clk
// B1 Update edt_update TK_upd
// B1 Bypass mode edt_bypass tap_i/tap_ctrl_i/edt_bypass_inst
// B1 Scan channel 1 input edt_channels_in1 TK_channel_in[0]
// B1 " " " output edt_channels_out1 TK_channel_out[0]
//
// B2 Clock edt_clock TK_clk
// B2 Update edt_update TK_upd
// B2 Bypass mode edt_bypass tap_i/tap_ctrl_i/edt_bypass_inst
// B2 Scan channel 1 input edt_channels_in1 TK_channel_in[1]
// B2 " " " output edt_channels_out1 TK_channel_out[1]
//
// ------------------------------------------------------------------
// Generic Connections
// ------------------------------------------------------------------
// Source Destination
// ------ -----------
// tap_i/tap_ctrl_i/test_en B1_inst/test_en
// tap_i/tap_ctrl_i/test_en B2_inst/test_en
//
```

Related Commands

add_edt_connections

delete_edt_connections

report_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

To report decompressors

report_edt_finder **-Decompressors** [-Type {ACtive | Inactive | ALl}]
[-Id ***decompressor_id#***] [-Filter ***instance_name_wildcard***]
[-Verbose [instances]] [{> | >>} ***file_pathname***]

To report scan chain segments

report_edt_finder **-Chain_segments**
[-Id ***decompressor_id#***] [-Type {ALl | ACcessible [COMpressed | Uncompressed | CONtrol] | Inaccessible}] [-MIn_length ***integer***] [-MAx_length ***integer***] [-Bins ***integer***]
[-Endpoints {***instance_name_wildcard*** | ***gate_id#***}]
[-Details {SUmmary | Endpoints | Full}] [{> | >>} ***file_pathname***]

To report masking logic

report_edt_finder **-Masking_logic** [-Id ***decompressor_id#***] [-Filter ***instance_name_wildcard***]
[-Verbose [instances]] [{> | >>} ***file_pathname***]

To report low-power shift controller logic

report_edt_finder **-Low_power_shift_controller** [-Id ***decompressor_id#***]
[-Filter ***instance_name_wildcard***] [-Verbose [instances]] [{> | >>} ***file_pathname***]

Description

Reports information about the EDT logic contained in a design.

Depending on the specified switches, the following columns can be reported:

- Length — Specifies the length of the identified scan chains. For example, (16-101) means the smallest scan chain has 16 cells and the longest scan chain has 101 cells.
- Count — Specifies the number of scan chains. For example, the first line of this report shows there are 40 chains that are compressed with length 16-101.

```
// Type         Length     Count
// ------------ -------- -------
// compressed   16-101     40
// compressed   102-188    50
// inaccessible 1-94       7520
// inaccessible 95-189     1160
```

- Type — Specifies the type of chain segment as defined by the Type argument description.
- Begin/End — Specifies the first scan cell or last scan cell of the scan chain segment. Defined by the -Details Endpoints switch.

Arguments

- **-Decompressors**

  A required switch that reports on decompressors found in the design.

- [-Type {ACtive | Inactive | ALl}]

  An optional switch and literal pair that specifies which decompressors to report on. Options include:

  ACtive — Reports on all active decompressors identified. This is the invocation default.

  Inactive — Reports on all inactive decompressors identified (decompressors that do not reach channel outputs).

  ALl — Reports all decompressors identified.

- **-Chain_segments**

  A required switch that reports on scan chain segments found in the design.

- -Type {ALl | ACcessible [Compressed | Uncompressed | CONtrol] | Inaccessible}}

  An optional switch and literal pair that specifies the type of chain segments to report. Literal options include:

  ALl — Reports all chain segments found. This is the invocation default.

  ACcessible — Reports all scan chain segments accessible via identified EDT logic or primary input/output pins. You can also enter an additional literal option to further sort on the accessible scan chains returned. Literal sort options include:

  COMpressed — Reports compressed chains, such as chains accessible via identified EDT logic.

  Uncompressed — Reports uncompressed chains such as chains accessible via primary input/output pins.

  CONtrol — Reports chain segments of the EDT control logic such as the mask shift registers of Xpress compactors.

  Inaccessible — Reports chain segments that are not accessible via identified EDT logic or primary input/output pins.

- -MIn_length ***integer***

  An optional switch and integer pair that specifies the minimum length of chain segments to display. The default is 1.

- -MAx_length ***integer***

  An optional switch and integer pair that specifies the maximum length of chain segments to display. The default is unlimited.

- -Endpoints {***instance_name_wildcard*** | ***gate_id#***}

  An optional switch and string or integer that specifies the endpoints of the chain segments to display. The string may include asterisk (*) and/or question mark (?) wildcard characters.

- -Details {SUmmary | Endpoints | Full}

  An optional switch and literal pair that specifies the details to display. Options include:

  SUmmary — Reports only a summary such as the number of chains that belong to a certain category. This is the invocation default.

  Endpoints — Reports the endpoints of the chain segments.

  Full — Reports all available information about the chain segments, including the instance name of each chain element.

- -Bins ***integer***

  An optional switch and integer pair that specifies the maximum number of bins that scan chain data is sorted by in the summary report.

  Each type (accessible/compressed/uncompressed/control/inaccessible) of scan chain reported is sorted by scan chain lengths determined by the number of bins specified with this switch. By default, 10 bins are used.

- **-Masking_logic**

  A required switch that reports details about specified Xpress mask registers identified by EDT Finder.

- **-Low_power_shift_controller**

  A required switch that reports details about low-power shift and hold registers identified by EDT Finder.

- -Id ***decompressor_id#***

  An optional switch and integer pair that specifies the identification number of the decompressor to report. The value of the *decompressor_id#* is the unique identification number automatically assigned to every decompressor by the EDT Finder.

- -Filter ***instance_name_wildcard***

  An optional switch and string pair that specifies the instance name of the decompressors or masking logic to report. You can use regular expressions that include asterisk (*) and/or question mark (?) wildcard characters.

- -Verbose [instances]

  An optional switch and literal that transcripts informational messages about the EDT logic found in a design. Depending on the EDT logic specified, this switch reports informational messages about decompressors, low power registers, and masking logic found in the design. If the *instances* literal is specified, only the names of instances are reported. By default, these messages are disabled.

- > ***file_pathname***

  An optional redirection operator and pathname pair that creates or replaces the contents of *file_pathname* with the output.

- >> ***file_pathname***

  An optional redirection operator and pathname pair that appends the output to the contents of *file_pathname*.

Example 1

The following example displays a summary of decompressors found in a design:

report_edt_finder -decompressors

```
// Decomp. Id# #Bits #Inputs #Chains  Block      Type
// ----------- ----- ------- -------  ---------  -----
//  1          16       4      28     m1_28x16   active
//  2          16       4       4     m2_4x32    active
//  3          16       4      50     m3_50x187  active
//  4          10       1       8     m4_8x16    active
```

Example 2

The following example displays a summary of all the masking logic found:

report_edt_finder -masking_logic

```
// Decomp. Id# Length Type     Block
// ----------- ------ ------   --------------
// 1           9      xpress   piccpu_40
// 2           31     xpress   my_core_5ch
// 3           31     xpress   my_core_5ch_2
```

Example 3

This example displays a summary of all found chain segments grouped into 2 bins per type:

report_edt_finder -chain_segments -bins 2

```
// Type          Length    Count
// ------------  --------  -------
// compressed    16-101    40
// compressed    102-188   50
// inaccessible  1-94      7520
// inaccessible  95-189    1160
```

Example 4

The following example displays the scan inputs and outputs of the accessible chain segments with a minimum length of 50:

report_edt_finder -chain_segments -type accessible compressed -min_length 50

```
//  EDT Control         Length  Type          Name
//  ----------------    ------  ------------  ------------------
//  edt controllable    50      compressed    piccpu_90_chain2
//  edt controllable    90      compressed    piccpu_90_chain3
//  edt controllable    50      compressed    piccpu_90_2_chain2
//  edt controllable    90      compressed    piccpu_90_2_chain3
```

Example 5

The following example displays a summary of the low-power shift controller logic in blk1:

report_edt_finder -low_power_shift_controller -id 2 -verbose instances

```
// Decomp. Id # #Registers     Block
// ------------ -------------  ---------------------
// 2            5              blk1
// Shiftreg.    /top/blk1_i/.../low_power_shift_reg_0_reg[4]/UD1(16171)
// Hold reg.    /top/blk1_i/.../control_hold_reg_0_reg[4]/DFF1(16166)
// Shiftreg.    /top/blk1_i/.../low_power_shift_reg_0_reg[3]/UD1(16172)
// Holdreg.     /top/blk1_i/.../low_power_hold_reg_0_reg[3]/DFF1(16167)
// Shiftreg.    /top/blk1_i/.../low_power_shift_reg_0_reg[2]/UD1(16173)
// Holdreg.     /top/blk1_i/.../low_power_hold_reg_0_reg[2]/DFF1(16168)
// Shiftreg.    /top/blk1_i/.../low_power_shift_reg_0_reg[1]/UD1(16174)
// Holdreg.     /top/blk1_i/.../low_power_hold_reg_0_reg[1]/DFF1(16169)
// Shiftreg.    /top/blk1_i/.../low_power_shift_reg_0_reg[0]/UD1(16175)
// Hold reg.    /top/blk1_i/.../low_power_hold_reg_0_reg[0]/DFF1 (16170)
```

Related Topics

set_edt_finder

report_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_edt_instances [-All_blocks] [-Errors_only] [{> | >>} ***file_pathname***]

Description

Displays the instance pathnames of the top-level EDT logic, decompressor, and compactor.

Note

For modular Tessent TestKompress, the report_edt_instances command operates only on the current EDT block (the EDT block that is the current target of EDT-specific commands). This is the EDT block specified by the last add_edt_blocks or last set_current_edt_block command, whichever you issued most recently.

The report_edt_instances command lists the names of the instances where the tool expects to find the top-level EDT logic, decompressor, and compactor. It also reports the pathnames of these instances when known. If the display lists any instance paths as "NOT FOUND," the tool will not be able to provide the maximum amount of detail in the debugging information for K19-K22 DRC violations. You can use the set_edt_instances command to update the instance information.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display instance information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- -Errors_only

  An optional switch that specifies to display instance inform &‡n only if the tool is unable to positively identify at least one of the instance names. This could be because the tool either cannot find the instance name/path, or the instance name is not unique.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example displays the current EDT instance data for a design, updates the information, then displays the updates:

report_edt_instances

```
// edt_logic_top
//    Setting: edt_i
//    Path   : NOT FOUND
//
// decompressor
//    Setting: edt_decompressor_i
//    Path   : NOT FOUND
//
// compactor
//    Setting: edt_compactor_i
//    Path   : NOT FOUND
```

**set_edt_instances -edt_logic_top my_core_edt_i **
**-decompressor my_core_edt_decompressor_i **
-compactor my_core_edt_compactor_i

```
// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
// Found compactor at path:
      "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
      "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

report_edt_instances

```
// edt_logic_top
//    Setting: my_core_edt_i
//    Path   : /my_core_edt_top/my_core_edt_i
//
// decompressor
//    Setting: my_core_edt_decompressor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//    Setting: my_core_edt_compactor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Related Commands

add_edt_blocks
set_edt_instances
set_current_edt_block

report_edt_lockup_cells

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_edt_lockup_cells [-All_blocks] [{> | >>} *file_pathname*]

Description

Displays a tabular report of the lockup cells in the EDT logic.

The report_edt_lockup_cells command reports details about the lockup cells in the EDT logic.

The command only reports lockup cells that will be included in the generated IP. Specifically, when lockup cell insertion is turned off (which is not recommended), the report does not include any lockup cells.

Note

During pattern generation, the command reports the expected lockup cells and not the actual lockup cells in the design.

The reported information is arranged in tables that cover the following:

Note

If you invoke this command in Setup mode because EDT DRC failed, some data will not be available and hence, will not be reported.

- Scan clocking—Details about the first and last scan cell in each scan chain. For each flip-flop (FF), the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO). The names of the first and last cell's clocks and their capture timing are also provided. The capture time is listed in the "Cap" column for the first cell, and in the "Chg" column for the last cell.
- Location—Each lockup cell is listed under one of the following categories depending on its location in the IP:
  - Decompressor—Lockup cells between the decompressor and the scan chain inputs
  - Compactor—Lockup cells between the scan chain outputs and the compactor
  - Bypass—Lockup cells in the bypass circuitry
- Type of lockup cell—Flip-flop (FF) or latch (LA). For each flip-flop, the report lists whether it is leading edge triggered (LE) or trailing edge triggered (TE). For each latch, the report lists whether it is active high (HI) or active low (LO).
- Clock—Name of the clock for each lockup cell

- Relevant Clocks—Summary of the off state, type, and timing of all clocks listed elsewhere in the report. The timing is based on the shift procedure in the test procedure file.

For more information, see "Understanding Lockup Cells" in the *Tessent TestKompress User's Manual*.

The -All_blocks switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the lockup cell information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

An optional switch that specifies to display the lockup cell information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays the information for all EDT blocks.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example reports lockup details for a design with eight scan chains and one scan channel. The EDT logic is FF-based and includes bypass circuitry:

set_system_mode analysis
report_edt_lockup_cells

```
// SCAN Clocking
//----------+-----------------------+-----------------------
//          |        First Cell     |       Last Cell
// Chain (#)|Gate (id) Type Cap Clock|Gate (id) Type Chg Clock
```

```
//----------+-----------------------+----------------------
// chain1(1)|/i7/ (92) FF-TE 20 /clk |/rg0/ (33) FF-TE 20 /clk
   ...
// chain8(8)|/i5/ (48) FF-TE 20 /clk |/ix8/ (89) FF-TE 20 /clk
//
// DECOMPRESSOR Lockup Cells
//----------+------------------+-----------------
//          | First cell       | Second cell
// Chain (#)| Type  Clock      | Type  Clock
//----------+------------------+-----------------
// chain1(1)| FF-TE /edt_clock | ----- ----------
// chain2(2)| FF-TE /edt_clock | FF-LE /edt_clock
   ...
// chain8(8)| FF-TE /edt_clock | ----- ----------
//
// COMPACTOR Lockup Cells
//----------+-----------------
//          | Lockup cell
// Chain (#)| Type  Clock
//----------+-----------------
// chain1(1)| FF-TE /edt_clock
   ...
// chain8(8)| FF-TE /edt_clock
//
// BYPASS Lockup Cells
//----------------------------+-----------
//                             |Lockup cell
// Chain  (#)  ->  Chain  (#) |Type  Clock
//-----Channel 1--------------+-----------
// chain1 (1)  ->  chain2 (2) |FF-TE /clk
// chain2 (2)  ->  chain3 (3) |----- -----
// chain3 (3)  ->  chain4 (4) |----- -----
// chain4 (4)  ->  chain5 (5) |----- -----
// chain5 (5)  ->  chain6 (6) |----- -----
// chain6 (6)  ->  chain7 (7) |----- -----
// chain7 (7)  ->  chain8 (8) |----- -----
//
// RELEVANT Clocks
//------------+-------+--------------+-----
//            |  Off  |              | Time
// Name       | State | Type         |LE TE
//------------+-------+--------------+-----
// /edt_clock |   0   | EDT          |20 30
// /clk       |   0   | Shift Master |20 30
```

The following is the bypass portion of a report for a design with eight scan chains and two scan channels. Notice that in addition to listing the bypass lockups, the bypass tables show the chain connections to the channels.

```
// BYPASS Lockup Cells
//--------------------------------+-----------
//                                |Lockup cell
// Chain    (#)  ->  Chain    (#)  |Type  Clock
//-----Channel 1------------------+-----------
// chain1   (1)  ->  chain2   (2)  |----- -----
// chain2   (2)  ->  chain3   (3)  |----- -----
// chain3   (3)  ->  chain4   (4)  |----- -----
```

```
// chain4   (4)  ->  chain5   (5)  |----- -----
// chain5   (5)  ->  chain6   (6)  |----- -----
//-----Channel 2-------------------+-----------
// chain7   (7)  ->  chain8   (8)  |----- -----
// chain8   (8)  ->  chain9   (9)  |----- -----
// chain9   (9)  ->  chain11 (10)  |FF-TE /NX1
// chain11 (10)  ->  chain10 (11)  |FF-TE /NX2
```

Example 2

The following example shows a design with eight scan chains; four of the scan chains are clocked by LE clk1 and the other four scan chains are clocked by TE clk2. The report_edt_lockup_cells output that follows shows the inserted lockup cells at the scan chain inputs and outputs.

set_edt_options -lockup on -retime_chain_boundaries on
set_system_mode analysis
report_edt_lockup_cells

```
//
//  SCAN Clocking
//  -----------+----------------------------------------+----------------------------------------
//             |                First cell              |                Last cell
//  Chain   (#) | Gate              (id) Type  Cap Clock | Gate             (id) Type  Chg Clock
//  -----------+----------------------------------------+----------------------------------------
//  chain1 (1) | /chain1_cell19/Q (234) FF-LE  40 /clk1 | /chain1_cell0/Q (215) FF-LE  40 /clk1
//  chain2 (2) | /chain2_cell19/Q (254) FF-LE  40 /clk1 | /chain2_cell0/Q (235) FF-LE  40 /clk1
//  chain3 (3) | /chain3_cell19/Q (274) FF-LE  40 /clk1 | /chain3_cell0/Q (255) FF-LE  40 /clk1
//  chain4 (4) | /chain4_cell19/Q (294) FF-LE  40 /clk1 | /chain4_cell0/Q (275) FF-LE  40 /clk1
//  chain5 (5) | /chain5_cell19/Q (314) FF-TE  50 /clk2 | /chain5_cell0/Q (295) FF-TE  50 /clk2
//  chain6 (6) | /chain6_cell19/Q (334) FF-TE  50 /clk2 | /chain6_cell0/Q (315) FF-TE  50 /clk2
//  chain7 (7) | /chain7_cell19/Q (354) FF-TE  50 /clk2 | /chain7_cell0/Q (335) FF-TE  50 /clk2
//  chain8 (8) | /chain8_cell19/Q (374) FF-TE  50 /clk2 | /chain8_cell0/Q (355) FF-TE  50 /clk2
//
//  ------  ---  ----------------  -----------  ------------
//                 Decompressor    Chain Input  Chain Output
//                    Lockup          Lockup       Lockup
//  Chain   (#)       Type  Clock  Type  Clock  Type  Clock
//  ------  ---  ----------------  -----------  ------------
//  chain1  (1)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain2  (2)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain3  (3)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain4  (4)  FF-TE /edt_clock  ----- -----  FF-TE /clk1
//  chain5  (5)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain6  (6)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain7  (7)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//  chain8  (8)  FF-TE /edt_clock  FF-LE /clk2  ----- -----
//
//  RELEVANT Clocks
//  -----------+-------+---------------+------
//             |  Off  |               |  Time
//  Name       | state | Type          | LE TE
//  -----------+-------+---------------+------
//  /edt_clock |   0   | EDT           | 40 50
//  /clk1      |   0   | Shift  MASTER | 40 50
//  /clk2      |   0   | System MASTER | 40 50
//
```

Related Commands

report_edt_configurations

set_edt_options

report_edt_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_retargeting, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_edt_pins [-All_blocks]
[-Group_by_pin_name]
[{> | >>} *file_pathname*]

Description

Displays the currently defined names and inversion status of EDT channel and control pins.

The report_edt_pins command displays the names of all EDT channel and control pins and whether the pin signals are inverted between the chip input pin and the EDT logic.

The -All_blocks switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. In this case, the command by default reports the information described above, but only for the EDT block you have specified as the current EDT block. If you include the -All_blocks switch, the command displays this information for every EDT block. You set the current EDT block using either the add_edt_blocks or set_current_edt_block command. "EDT block" is the term used for a decompressor/compactor pair and the scan chains it controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks. If you are using a standard (non-modular) Tessent TestKompress flow, you do not need this switch.

  An optional switch that specifies to display the EDT channel and control pin information for all EDT blocks. Without this switch, the tool displays information for only the current EDT block, if it is defined. If a current EDT block is not defined, the tool displays configuration information for all EDT blocks.

- -Group_by_pin_name

  An optional switch that specifies to display the pin name and all of the blocks that share that pin.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

Assume the tool is in setup mode. The following example reports, for a design with two scan channels and no shared EDT pins, the names and inversion status of the EDT pins. The example then shares the EDT clock pin with a functional pin called “a1”. Following that, it renames the dedicated EDT bypass pin from the default name to “my_bypass” and specifies in the same command that the bypass signal is inverted between the chip input pin and the EDT logic. Afterward, the example again reports the EDT pins.

report_edt_pins

```
//
//    Pin description         Pin name           Inversion
//    ---------------         --------           ---------
//    Clock                   edt_clock              -
//    Update                  edt_update             -
//    Bypass mode             edt_bypass             -
//    Scan channel 1 input    edt_channels_in1       -
//     "      "    " output   edt_channels_out1      -
//    Scan channel 2 input    edt_channels_in2       -
//     "      "    " output   edt_channels_out2      -
//
```

set_edt_pins clock a1
set_edt_pins bypass my_bypass -inv
report_edt_pins

```
//
//    Pin description         Pin name           Inversion
//    ---------------         --------           ---------
//    Clock                   a1                     -
//    Update                  edt_update             -
//    Bypass mode             my_bypass              inv
//    Scan channel 1 input    edt_channels_in1       -
//     "      "    " output   edt_channels_out1      -
//    Scan channel 2 input    edt_channels_in2       -
//     "      "    " output   edt_channels_out2      -
//
```

Example 2

The following example reports each pin name and lists all of the blocks that share that pin. Channel sharing information is shown in red font.

report_edt_pins -all -group_by_pin_name

```
Pin name     Block  Pin                 Inversion  Channel pipeline stages
                    Description                   (mask register)
=================================================================
edt_clock    A      Clock               -           -
             B      Clock               -           -
             C      Clock               -           -
edt_update   A      Update              -           -
```

```
                B       Update              -           -
                C       Update              -           -
    channel_in_A A      Scan channel 1      -           7
                        input(control)
    channel_in_B B      Scan channel 1      -           "
                        input(control)
    channel_in_C C      Scan channel 1      -           "
                        input(control)
    channel_in1  A      Scan channel 2      -           7
                        input
                B       Scan channel 2      -           "
                        input
                C       Scan channel 2      -           "
                        input
    channel_in2  A      Scan channel 3      -           7
                        input
                B       Scan channel 3      -           "
                        input
                C       Scan channel 3      -           "
                        input
    channel_outA A      Scan channel 1      -           0
                        Output
    channel_outB B      Scan channel 1      -           "
                        Output
    channel_outC C      Scan channel 1      -           "
                        Output
```

Related Commands

report_drc_rules
report_edt_configurations
report_scan_volume
set_edt_options
set_edt_pins
write_edt_files

report_environment

Context: unspecified, all contexts

Mode: all modes

Usage

report_environment [{> | >>} *file_pathname*]

Description

Displays the current value(s) of many of the most common “set_...” commands.

When you first invoke the tool, the values displayed are the default settings for these commands.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example

The following example reports the current conditions under which the tool tests the circuit, then performs an ATPG run:

set_system_mode analysis
add_faults -all
report_environment
create_patterns

The output from the report_environment command may look like the following:

```
abort limit =              30
atpg compression =         OFF
atpg limits =              none
bist initialization =      X
bus simulation method =    global
capture clock =            none
checkpoint =               OFF
clockpo patterns =         ON
clock restriction =        clock_po
clock_off simulation =     OFF
contention check =         ON, mode = bus, handling = warning
DRC transient detection = ON -Verbose
fails report =             OFF
fault mode =               uncollapsed
fault type =               stuck
gate level =               design
gate report =              normal
```

```
iddq checks =             none, handling = warning
iddq strobe =             label
learn reporting =         OFF
logfile handling =        OFF
net dominance =           wired-gate
net resolution =          wired-gate
observation point =       master
pattern classification =  ON
pattern source =          internal
possible credit =         50%
pulse generators =        ON
RAM initialization =      uninitialized
RAM test mode =           static_pass_thru
random atpg =             ON
random clocks =           none
random patterns =         1024
screen display =          ON
shadow checking =         ON
simulation mode =         combinational    depth = 0
skew load =               OFF
split capture cycle =     OFF
stability check =         ON
system mode =             setup
tester cycles =           PLL_capture 10 timeplate_1
TLA loop handling =       OFF
trace report =            OFF
Z handling =              int=X  ext=X
Zhold behavior =          ON
```

Related Commands

All SET commands

report_pattern_filtering

report_external_simulator

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

Mode: setup, analysis

Usage

report_external_simulator [{> | >>} *file_pathname*]

Description

Displays the shell command the tool will use to invoke an external simulator for verifying patterns.

The report_external_simulator command displays the current setting of the set_external_simulator command. You use the latter to specify the shell command the tool should use to invoke an external time-based simulator for verifying patterns.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example checks the current external simulator setting, then specifies for the tool to invoke ModelSim in batch mode (-c) with loading messages disabled (-quiet), when verifying simulation results:

report_external_simulator

```
…/vsim -c
```

set_external_simulator vsim -c -quiet
report_external_simulator

```
…/vsim -c -quiet
```

Related Commands

set_external_simulator

report_failures

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

Mode: analysis

Usage

report_failures [{*pin_pathname* -Stuck_at {0 | 1}}…]
[-Pdet | -Exact] [-Max *integer*] [-Cycle]
[-MEasure_scan_at {CHANnels | CHAIns}]
[{> | >>} *file_pathname*]

For “patterns -failure_mapping”

report_failures -UNMapped

Description

Simulates the current design and test patterns and reports the failures in a pattern-based format.

For failure mapping, reports the unmapped top-level failures.

Arguments

- *pin_pathname* -Stuck_at {0 | 1}

  An optional, repeatable string, switch and literal triplet that specifies both the location and the value of a stuck-at fault to check for failing patterns. A fault simulation is run on the specified fault and the results are reported. Argument options include:

  pin_pathname — A string that specifies the pin pathname of the fault whose failing patterns you want to identify.

  -Stuck_at 0 | 1 — A switch and literal that specify the stuck-at fault value to simulate. The stuck-at options include:

  0 — stuck-at-0

  1 — stuck-at-1

- -Pdet

  An optional switch that directs the tool to report a failure when an X is simulated where a binary value is expected. By default, only the binary detections are reported.

- -Exact

  An optional switch directs the tool to report a failure when an X is simulated where a binary value is expected (like the -Pdet switch) and also when a binary value is simulated where an X is expected. By default, only the binary detections are reported.

- -Max *integer*

  An optional switch and integer pair that specifies a maximum number of failing patterns before the command stops the simulation. By default, all failing patterns are simulated.

- -CYCle

  An optional switch that reports the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the "Input File Requirements" chapter in the *Tessent Diagnosis User's Manual*.

- -measure_scan_at {channels | chains}

  An optional switch and literal pair that instructs the tool to report the failing bits of an EDT-enabled design at the scan elements; normally, the tool reports the failing bits at the EDT external channels. For an EDT-enabled design, this switch is only valid for external pattern sets in ASCII or binary format; STIL and WGL formats are not supported.

  If EDT is enabled (On), by default, the tool measures at the external channels so that all failing bits are compressed. If you specify the chains option in this case, the tool reports the failing bits at the scan chain cell_number without considering the output compaction logic.

  If EDT is disabled (Off), the tool always measures at the scan cells. In this case, specifying the channels option creates an error condition.

  Reporting failures using the chains option generates the following information:

  pattern_id — pattern index where the failure occurs.

  chain/PO_name — scan chain or the primary output name where the failure occurs.

  cell_number — scan cell index where the failure occurs. This field will be empty if the failure occurs at a primary output.

  expected_value — expected value observed at the scan chain output pin from the external pattern set for the failing point.

  simulated_value — the simulated value from the internal simulation for the failing point.

  cell_path_name — cell or primary output name of the failing point.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

- -unmapped

  A required switch that specifies to report all failures that Tessent Diagnosis could not map to core instances.

Example 1

The following example loads a test pattern source, runs simulation, and displays the pattern-based failure information.

set_system_mode analysis
read_patterns file1
report_failures

```
// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec
```

The pattern-based failure data is written in columns from left to right: pattern_id, chain/PO_name, cell_number, expected_value, simulated_value, cell_path_name as shown in the following example.

```
chain_test
0 chain5   0   H  L  //  uUART/reg_MODE0_CLK
0 chain5   1   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_2_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   5   H  L  //  uINTR/ix546
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   16  H  L  //  uINTR/ix236
0 chain5   88  H  L  //  padding_cycle
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  0
scan_test
0 chain5   2   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_3_
0 chain5   4   H  L  //  uCNTR/reg_PORT_3_SAMPLE_P1_5_
0 chain5   6   H  L  //  uINTR/ix106
0 chain5   8   H  L  //  uINTR/ix476
0 chain5   9   H  L  //  uINTR/ix446
0 chain5   11  H  L  //  uINTR/ix386
0 chain5   12  H  L  //  uINTR/ix356
0 chain5   13  H  L  //  uINTR/ix326
0 chain5   28  H  L  //  uSTM/reg_WAIT_SYNC/SFFR
0 chain5   100 H  L  //  padding_cycle
last_pattern_tested  2
```

Example 2

The following example returns a list of the top-level failures for core instance logic_die_BOT_inst__logic_die__internal.

report_failures -unmapped

```
//Note: TestSuite[1] '../data/top_flog1',  #fail_bits=1

Cycle  Pin                  Expected  Actual

688    bc_2_core_2_q1       L         H
```

Example 3

The following example shows the output of the report_failures command when the “–measure_scan_at chains’ switch and option combination is specified for an EDT-enabled design:

FAULT> read_patterns patterns.ascii
FAULT> report_failures –measure_scan_at chains

```
// pattern_id chain/   cell   expected simulated cell_path_name
//            PO_name  number value    value
   0          chain1   2      H        L         // all_rams_i/sf_0003_1
   0          chain4   3      H        L         // all_rams_i/sf_0028_1
   0          chain10  1      L        H         // all_rams_i/lf_0055_1
   0          chain11  5      L        H         // all_rams_i/lf_0043_1
   1          chain7   4      H        L         // all_rams_i/sf_0053_1

// incorrect_patterns=2, simulated_patterns=61, simulation_time=0.04sec.
```

Related Commands

diagnose_failures
read_failures
read_patterns
write_failures

report_false_paths

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

Mode: setup, analysis

Usage

report_false_paths [-All] [-Error] [-Verbose] [-Debug_error]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]
| [-GAte {{*gate_ID#* | *pin_pathname*}...}] ...
[{> | >>} *file_pathname*]

Description

Displays false paths previously defined with the add_false_paths command.

Note that you can display SDC filenames and line numbers by first issuing the "set_timing_exceptions_handling -line_no" command.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) false paths are reported.

Arguments

Tip: When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. An asterisk is treated as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that displays all previously-defined false paths. This is the default.

- -Error

  An optional switch that displays only the false paths that have errors and warnings.

- -Verbose

  An optional switch that displays all the gate information in the from, to, and through gate lists.

- -Debug_error

  An optional switch that displays all the previously-defined false paths that have errors and warnings, along with their respective error and warning messages.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a false path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI) or an internal clock pin name. For an instance pathname, all the output pins of the instance are

considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a false path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

Note

For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_false_paths... -to CLK2” would report false paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a false path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal, and the data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the false path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, their order left-to-right in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following false path specification:

  add false paths -from A -through B C -through D E -to F

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be false paths.

- -GAte *gate-ID# | pin_pathname*

  An optional repeatable switch and repeatable integer or string that filters the report results based on gates or pins that you specify. The tool reports only the false paths and multicycle paths that include the specified gates or pins as the definition points (-from, -to, or -through). You can use multiple -Gate switches to specify additional pins or gates in a single command invocation.

  gate-ID# — A repeatable integer that specifies the gate identification number. When you specify multiple gate IDs, the command reports on any path that includes any of those gates as definition points.

  pin_pathname — A repeatable string that specifies the name of a pin within the design.When you specify multiple pin_pathnames, the command reports on any path that includes any of those pins as definition points.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example displays all previously-defined false paths.

report_false_paths -all

```
// False Path -from LINESTATE[1] LINESTATE[0] -to_clock MCLK
// False Path -from_clock UTMICLK[0] -to_clock MCLK
//
// ----------------------------------------------
// The false path below has an error.
// ----------------------------------------------
// False Path -from dummy1 -to dummy2
//
// ----------------------------------------------
// The false paths below have warnings.
// ----------------------------------------------
// False Path -from_clock S -to_clock s
// False Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN
// False Path -from CLK1 -through U7/CP CLK1
//
// Total paths reported = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//          Use the command "report_false_paths -debug_error" to report
//          the causes of the errors/warnings.
// Warning: False path defined through all primary inputs, including all
//          clocks, which prohibits at-speed testing. Deleted clocks from
```

```
//          this list.
// Warning: There exists a false path that prohibits at-speed testing of
//          clock domain /CLK1.
//          This may result in lower test coverage.
```

Example 2

The following example displays the false paths with error and warning messages.

report_false_paths -error

```
// Error: False Path -from dummy1 -to dummy2
// Warning: False Path -from_clock S -to_clock s
// Warning: False Path -from CLK1 -through U7/CP CLK1
// Warning: False path -through I1 I2 I3 I4 I5 SIN CLK1 SEN
// Total reported false paths = 4
```

Example 3

The following example displays debug information for the false paths with error and warning messages.

report_false_paths -debug_error

```
// Error: False Path -from dummy1 -to dummy2 did not find gates dummy1
//        dummy2.
// Warning: False Path -from_clock S -to_clock s has no combinational path
//          between start and end points.
// Warning: False Path -from CLK1 -through U7/CP CLK1 prohibits at-speed
//          testing of clock domain /CLK1.
// Warning: False path -through I1 I2 I3 I4 I5 SIN CLK1 SEN defined
//          through all primary inputs, including all clocks, which
//          prohibits at-speed testing. Deleted clocks from this list.
// False path debug summary: 1 error, 3 warnings
```

Example 4

The following example displays the false paths that start at /my_design/d_in[6]. Notice the empty “()” for the incorrectly defined path.

report_false_paths -verbose -from /my_design/d_in[6]

```
// False path -from /my_design/d_in[6] ( /reg_d_1_7_/Q (81)
// /reg_d_1_6_/Q (73) ) -to /my_design/u25 ( /my_design/u25/Y (130) )
//
// ----------------------------------------
// The false path below has an error.
// ----------------------------------------
// Error: False path -from /my_design/d_in[6]( ) -to /my_design/xray[5]( )
//
// Total reported false paths = 2
// Warning: The last 1 path reported has an error.
//          Use the command “report_false_paths -debug_error” to report
//          the cause of the error.
```

Example 4

The following example displays the false paths and multicycle paths whose definition points (-from, -to, or -through) include the gate top/inst1/A:

```
report_false_paths -gate top/inst1/A
```

Example 5

The following shows false path reporting after turning on reporting of SDC filenames and line numbers:

```
read_sdc timing.sdc
set_timing_exceptions_handling -line_no on
report_false_paths

//  False Path -from_clock cmem_fclki cmem_fclkin -to_clock p_clk spm_pci_clk \
-setup  (filename = timing.sdc, line 406)

//  False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclki cmem_fclkin \
-setup  (filename = timing.sdc, line 408)

...

//  Total reported false paths = 177
```

Related Commands

add_false_paths
delete_false_paths
delete_multicycle_paths
read_sdc
report_multicycle_paths
set_timing_exceptions_handling

report_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Faults Usage:

report_fault_sites [-All | *path_name*] [-Path *gate_id_begin gate_id_end*]
[-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]
[{> | >>} *file_pathname*]

Bridge Faults Usage:

report_fault_sites
{[-All | *net_pathname_pair* | *bridge_name*… | *net_pathname*… | *instance_name*…]
[-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-ATtribute] [-NOEQ]} |
[-Histogram {Layer | Type | {Weight [-Sets *number*]}}]
[-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]
[{> | >>} *file_pathname*]

UDFM Usage:

report_fault_sites [-ALl | instance_name] [-Udfm_type *name*][-CEll *name*]
[-MOdule *name*] [-FAult *name*] | { [-UNDEFined_cells] [-DEFined_cells] }
[{> | >>} *file_pathname*]

Description

Depending on the fault model, displays path definitions, bridge entries, or UDFM entries from the internal list.

The report_fault_sites command displays the definitions in the tool's internal list for the specified faults. Fault definitions are loaded into the internal list with the read_fault_sites command.

For more information about bridge faults, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information about path delay faults, refer to "Creating a Path Delay Test Set" in the *Tessent Scan and ATPG User's Manual*.

For more information about UDFM faults, refer to "User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

Path Delay Specifics

The report_fault_sites command also displays unambiguous paths you previously added to the internal path list using the add_ambiguous_paths command. When displaying derived unambiguous paths, report_fault_sites cannot list a gate if it does not have a pin pathname associated with it. For some designs, gates without associated pin pathnames could be fairly common. Therefore, refer to the displayed unambiguous paths for launch and capture points, but be aware the display may not show all the gates in between.

Note

If you get ambiguous path warnings, then add_ambiguous_paths, those paths when written out lose the information about which of the ambiguous paths they were. Consequently, you cannot reload them and get back to the same set of paths.

UDFM Specifics

The report_fault_sites command doesn't report restricted information about UDFM models generated by Tessent CellModelGen, which protects the IP inside of those models.

Arguments

- -All

  An optional switch that displays all fault definitions for the currently loaded paths, bridge, or UDFM entries. This is the default.

- *path_name*

  An optional string that specifies the name of a path whose definition you want to display. The path must be one defined in a path definition file that you previously loaded using the read_fault_sites command.

- -Path *gate_id_begin gate_id_end*

  An optional switch and two-integer triplet that specifies a particular path or portion of a path whose definition you want to display. Use this argument to report on paths that were not defined in a path definition file, and therefore were not loaded using the read_fault_sites command.

  The two integers specify two gate identification numbers that indicate the beginning and ending of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.

  The value of *gate_id_begin* or *gate_id_end* is the unique identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- *net_pathname_pair*

  A string pair that specifies the pathname to the nets or pins in a bridge entry. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  A repeatable string that specifies a bridge name.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname.

- *instance_name*

  A repeatable string that specifies an instance pathname.

- -ATtribute

  An optional switch that displays the attributes of the reported bridge entries. Attributes may include distance, weight, layer, and parallel_run.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

  If the string is an instance that contains a non-dominant net from a bridge entry, a fault is reported for each bridge entry associated with the instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- -NOEQ

  An optional switch that reports only the representative fault information: the net pair and fault categories.

- -Histogram {Layer | Type | {Weight [-Sets *number*]}

  An optional switch and literal pair that displays a histogram of the number of equivalent classes of bridges based on either their layer, type, or weight attribute. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying “-sets 0” forces the tool to display an individual line for each bridge of different weight.

  The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.

- -Display {[DEBug] [DESign] [DAta] | ALl} [APpend]
  An optional switch, literal, and literal that displays the reported information graphically in the specified DFTVisualizer window(s); if *Append* is specified, the information is added to the existing contents of the window instead of replacing the contents. The choices are as follows:

  DEBug — Debug window

  DESign — Design window

  DAta — Data window

  ALl — A literal that displays the information in all of the preceding windows.

- -Udfm_type *name*

  An optional switch and string pair that specifies the name of a UDFM fault type. Use this option to display the faults associated with the specified UDFM fault type. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -CEll *name*

  An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -MOdule *name*

  An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with the specified module. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -FAult *name*

  An optional switch and string pair that specifies the name of a UDFM defined fault. Use this option to display all faults associated with the specified name. The name string can include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -UNDEFined_cells

  An optional switch that reports cells without a valid UDFM definition that were added with a previous read_fault_sites command. This switch works only with an instance_name.

- -DEFined_cells

  An optional switch that reports cells with a valid UDFM definition that were added with a previous read_fault_sites command. This switch works only with an instance_name.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following path delay example reads in (loads) the path information and then displays that data:

set_fault_type path_delay
read_fault_sites /user/design/pathfile
report_fault_sites

```
PATH "path0" =
   PIN /I$6/Q + ;
   PIN /I$35/B0 + ;
   PIN /I$35/C0 + ;
   PIN /I$1/I$650/IN + ;
```

```
   PIN /I$1/I$650/OUT - ;
   PIN /A_EQ_B + ;
END ;
PATH “path1” =
   PIN /I$6/Q + ;
   PIN /I$35/B1 + ;
   PIN /I$35/C1 + ;
   PIN /I$1/I$649/IN + ;
   PIN /I$1/I$649/OUT - ;
   PIN /I$5/D - ;
END ;
```

Example 2

The following bridge example reads the bridge information in the bridge fault definition file, */user/design/bridgefile*, then displays the data for bridge entries added to the fault site list:

set_fault_type bridge
read_fault_sites /user/design/bridgefile

```
//
//  Load bridge entries from file "/user/design/bridgefile".
//
//  5 bridge entries were read from file "/user/design/bridgefile".
//  2 bridge entries were skipped due to unknown net name.
//  3 bridge entries were added to the fault site list.
```

report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  NAME = "U0.BRIDGE";
}
```

Example 3

The following example displays the attribute data for the “U0.BRIDGE” bridge entry:

report_fault_sites U0.BRIDGE -name -attribute

```
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {-, -, -, -};
  WEIGHT = 1.0000;
```

```
  DISTANCE = 4.93827000e-06 um;
  PARALLEL_run = 6.07500000e-05 um;
  X_COORDINATE = 152500;
  Y_COORDINATE = 739500;
  LAYER = "metal1";
  NAME = "U0.BRIDGE";
}
```

Example 4

The following example displays the UDFM fault definition f01 defined in UDFM type my-udfm-sa-faults of instance /top/inst1:

report_fault_sites /top/inst1 -udfmtype my-udfm-sa-faults -fault f01

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

If there is more than one UDFM types used, the following command reports all f01 faults on instance /top/inst1:

report_fault_sites /top/inst1 -Fault f01

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
   UDFMType("intra-cell") {
      Instance("/top/inst1") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 418
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

report_fault_sites -all

```
UDFM {
   Version:1;
   UdfmType("my-udfm-sa-faults") {
      Cell("AOX") {
         // Source reference:
         // File: /home/JohnDoe/myadded.udfm, Line: 145
         Fault("f01") {
            Test { StaticFault{"Z":0;} Conditions{"D":0;"E":1; "F":-;} }
            Test { StaticFault{"Z":1;} Conditions{"D":1;"E":-; "F":0;} }
         }
      }
   }
}
```

Example 5

The following example shows how to display a list of cells both with and without valid UDFM definitions:

report_fault_sites -defined_cells

```
//  Report of cells defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  XNOR2X18         518
//  XNOR2X3          139
//  XOR2X17          387
//  AND2X35          913
//  AO112X27           4
//  AO12X27          203
//  AO22X27          219
//  AO312X27          39
//  AO31X27            5
//  ---------  ---------
```

report_fault_sites -undefined_cells

```
//  Report of cells not defined by UDFM
//
//  Cell Name  Instances
//  ---------  ---------
//  cnhlsx4v          52
//  df1qx1v          834
//  df4sqx1v           7
//  u_noti            68
//  u_mux2          7261
//  u_mux3            20
//  u_mux4           398
//  ---------  ---------
```

Related Commands

add_ambiguous_paths
add_fault_sites
add_faults
delete_fault_sites
read_fault_sites
set_fault_type
write_fault_sites

report_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Usage: report_faults [-Both | -Rise | -Fall] [-VErilog | -VHdl]
[-All] [-Class *fault_class*[.*fault_subclass*]] [*object_pathname*] [-NO_Subclass]

Bridge Usage: report_faults { [-Histogram Weight [-Sets *number*]] }
| { [-Histogram { Layer | Type }] }
| { [-ALl | *net_pathname_pair* | { *bridge_name* ... -NAME }
| { *net_pathname* ... -SINGle } | { *instance* ... -SINGle }]
[-Class *fault_class*[.*fault_subclass*]] [-NET_pathname | -PIN_pathname] [-NOEQ]
[-NO_Subclass] [-VErilog | -VHdl] [-ATtribute] [-SCAN_Enable] [-CLOCK_Cones]
[-IO] [-ASYnchronous_controls] }
| { [-CLOCK_Domains { ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] } [(> | >>) *file_name*]

Toggle/Iddq Usage: report_faults [-Stuck_at [01 | 0 | 1]] [-NO_Subclass]
{ [-Class *fault_class*[.*fault_subclass*]] [*object_pathname* ...] [-NOEQ]
[*instance_name* ...] [-Hierarchy *integer*] [-Min_count *integer*] [-VErilog | -VHdl]
[-All | -Protected_only | -Unprotected_only] [-CEll_name] [-CLOCK_Domains
{ ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] }
| { [-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls] }
| { -UNlisted } [(> | >>) *file_name*]

Stuck/Transition Usage: report_faults [-Stuck_at [01 | 0 | 1]] [-NO_Subclass]
{ [-Class *fault_class*[.*fault_subclass*]] [*object_pathname* ...] [-NOEQ]
[*instance_name* ...] [-Hierarchy *integer*] [-Min_count *integer*] [-VErilog | -VHdl]
[-All | -Protected_only | -Unprotected_only] [-CEll_name]
[-CLOCK_Domains { ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] [-DElay_data]
[-TIMING_CRITICAL] } | { [-SCAN_Enable] [-CLOCK_Cones] [-IO]
[-ASYnchronous_controls] } | { -UNlisted } [(> | >>) *file_name*]

UDFM Usage : report_faults [-NO_Subclass] { -All | [-Class *fault_class*[.*fault_subclass*]]
| *instance_pathname* | [-Hierarchy *integer*] [-NOEQ]
[-CLOCK_Domains { ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]] { [-Udfm_type *name*]
[-INstance *name*] [-CEll *name*] [-MODule *name*] [-SOurce] }}
| { -UNlisted } [(> | >>) *file_name*]

Power-Aware Options: (applicable only after you've loaded CPF/UPF power data)
[[ON_domains] | [-OFf_domains] | [-POWer_domains {*domain_name* ...}]]
[-ISolation_cells] [-LEvel_shifters] [-REtention_cells]

Description

Displays fault information from the current fault list.

The report_faults command displays faults you added to the fault list using the add_faults or read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any arguments, report_faults displays information on all the known faults.

By default, the report_faults command ignores the constraint values implied in the data path when reporting faults in the data path. However, the tool still considers the constraint values for clock path faults.

For more information about UDFM faults, refer to "User-Defined Fault Modeling" in the *Tessent Scan and ATPG User's Manual*.

The power-aware options report_faults based on power domains and/or power features after you have loaded a CPF or UPF file.

The report_faults command displays the following columns of information for each fault:

- fault value - The fault value may be either 0 (for stuck-at-0 or "slow-to-rise" transition faults) or 1 (for stuck-at-1 or "slow-to-fall" transition faults).
- fault code - A code name that indicates the lowest level fault class assigned to the fault (see Table 7-26). This also includes a fault sub-class code when available (see Table 7-50 on page 1921).
- fault site - The pin pathname of the fault site.
- cell name (optional) - The name of the cell corresponding to the fault (displayed only if you include the -Cell_name argument).

You can use the -Hierarchy option to display a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option which specifies the minimum number of faults that must be in a hierarchical level before they are displayed.

You may choose to display either collapsed or uncollapsed faults by using the set_fault_mode command. Also, some fault data is large and it would be more appropriate to use the write_faults command, and then read the file contents.

Note that this command can report specific AU.TC and AU.PC faults when you previously issue the "report_statistics -detailed_analysis" command. For AU.TC faults you must specify gate IDs rather than the full pin pathname. For AU.PC faults, you must specify the full pin pathname. For more information, refer to "Example 9" on page 1334.

Arguments

- -Class *fault_class*[.*fault_subclass*]

  An optional switch and literal pair that specifies the fault class or fault class/sub-class to display. Table 7-26 lists the valid fault class codes and their associated fault class names. Table 7-50 on page 1921 lists the valid fault sub-class names. You should use the code when specifying a fault class or sub-class.

Table 7-26. Fault Class Codes and Names

Fault Class Code	Fault Class Name	Fault Class Coverage
FU	Full	TE+UT
TE	TEstable	DT+PD+AU+UD
DT	DETEcted	DS+DI+DR
DS	DET_Simulation	
DI	DET_Implication	
DR	DET_Robust (Path Delay Testing only)	
DF	DET_Functional (Path Delay Testing only)	
PD	POSDET	PT+PU
PU	POSDET_Untestable	
PT	POSDET_Testable	
AU	Atpg_untestable	
UD	UNDetected	UC+UO
UC	UNControlled	
UO	UNObserved	
UT	UNTestable	UU+TI+BL+RE
UU	UNUsed	
TI	TIed	
BL	Blocked	
RE	Redundant	

- -NO_Subclass

  Prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when available.

- -UNlisted

  Displays unlisted fault data. If there are multiple grayboxed instances in the design, this switch reports accumulated fault numbers.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition faults you want to display. The choices are as follows:

  01 — A literal specifying that for stuck-at faults the tool display both “stuck-at-0” and “stuck-at-1” faults; or for transition faults, the tool display both “slow-to-rise” and “slow-to-fall” faults. This is the default.

  0 — A literal specifying to display only the “stuck-at-0” faults (“slow-to-rise” faults for transition faults).

  1 — A literal specifying to display only the “stuck-at-1” faults (“slow-to-fall” faults for transition faults).

- -All

  An optional switch that displays all of the faults, protected as well as unprotected, on all model, netlist primitive, and top module pins.

- *object_pathname*

  An optional repeatable string that specifies a list of pins, instances, or delay paths whose faults you want to display.

- -Hierarchy *integer*

  An optional switch and integer pair that specifies the maximum hierarchy level for which you want to display a summary of the faults. However, note that the reported number of faults depends on other switches, like -class.

- -Min_count *integer*

  An optional switch and integer pair that you can use with the -Hierarchy option to specify the minimum number of faults that must be in a hierarchical level to display the hierarchical summary. The default is 1.

- -NOEQ

  - For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that turns off the display of “EQ” as the fault class for any equivalent faults; the fault class displayed is then that of the representative fault. When you do not specify this switch, the tool displays an “EQ” as the fault class for any equivalent faults. For more information on representative and equivalent faults, see “Fault Collapsing” in the *Tessent Scan and ATPG User’s Manual*.

  - For bridge faults only, an optional switch that reports only the representative fault information: the net pair and fault categories.

- -Both | -Rise | -Fall

  An optional switch that specifies which faults to display for each path already added via the Add Paths command. These switches are used for path delay faults only.

  -Both - An optional switch that specifies to display both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise - An optional switch that specifies to display only the slow-to-rise faults.

  -Fall - An optional switch that specifies to display only the slow-to-fall faults.

- -VErilog | -VHdl

  An optional switch pair that output the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist independent format.

- -Protected_only

  An optional switch that displays only the protected faults on all model, netlist primitive, and top module pins.

- -Unprotected_only

  An optional switch that displays only the unprotected faults on all model, netlist primitive, and top module pins.

- -CEll_name

  An optional switch that lists the name of the corresponding circuit element (ATPG library cell, Verilog primitive, primary input or primary output) for each fault as listed in Table 7-27. Cell names for equivalent faults are based on the cells associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 7-27. Name Conventions Used by report_faults -Cell_name

Corresponding Circuit Element	Listed Cell Name
ATPG library cell	Name of the ATPG library cell
Supported Verilog primitive (for a list, see Table 4-1 in the *Tessent Cell Library Manual*)	Name of the Verilog primitive
Primary input or output	“primary_input” or “primary_output”

- -CLOCK_Domains {ALL | *clock_pathname*…}

  An optional switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to report only faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock. The argument choices are as follows:

  ALL — A literal that specifies all the clocks in the design.

clock_pathname — A repeatable string that specifies a particular clock.

Be aware that a fault reported using this switch might also be detectable by an unspecified clock.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from reporting faults in equivalent clock domains.

- -INCLUDE_INTER_NORACE_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to take user-defined non-race clocks into account when reporting faults in the clock domain. You can use this switch in non-setup mode and when the clock restriction is set to be domain_clock.

- -CAPture_procedures {ALL | *capture_procedure_name*…}

  An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to report faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  See the set_capture_procedures command description for information about enabling or disabling named capture procedures.

- -DElay_data

  An optional switch that reports the fault list including the slacks of the faulty site. The reported slack is for information only and cannot be loaded back into the system.

- -TIMING_CRITICAL

  An optional switch that reports timing-critical faults.

Note

In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See "Using Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

- -SCAN_Enable

  An optional switch that specifies to report faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch reports all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that specifies to report all faults that only fan out to clock ports, Set/Reset ports or Read/Write control ports of memory elements, including RAM and ROM. Notice that this applies to any memory elements, not just scan cells.

- -IO

  An optional switch that specifies to report faults that are:

  - Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
  - Only observed by POs

- -ASYnchronous_controls

  An optional switch that specifies to report all faults that only fan out to Set/Reset ports of state elements and RAMs. This switch applies to a subset of the faults the -CLOCK_Cones switch reports.

- -Histogram Weight [-Sets *number*]
  -Histogram { Layer | Type }

  An optional switch and literal pair that displays a histogram of the number of equivalent classes of bridges based on either their layer or type, or their weight attributes. When you specify a weight histogram, you can optionally include the -Sets switch to specify a number of sets into which to distribute the report data. Specifying “-sets 0” forces the tool to display an individual line for each bridge of different weight.

  The display for a weight histogram consists of one weight histogram per type of bridge fault, whereas a layer or type histogram is a single histogram.

- > *file_name*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_name*.

- >> *file_name*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_name*.

- *net_pathname_pair*

  An optional, string pair that specifies the bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  An optional, repeatable string that specifies the bridge name.

- *net_pathname*

  An optional, repeatable string that specifies a single net pathname.

- *instance_name…*

  An optional, repeatable string that specifies an instance pathname.

- -ATtribute

  An optional switch that specifies to display the attributes for bridge entries. Attributes may include distance, weight, layer, and parallel_run.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net or instance. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

  If the string specifies an instance that contains a non-dominant net from a bridge entry, a fault is reported for each bridge entry associated with the instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- *instance_pathname …*

  An optional repeatable string that specifies a list of instance names you want to display.

- -Udfm_type *name*

  An optional switch and string that specifies the name of a UDFM fault type. Use this option to display the faults associated with a specified UDFM fault type. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -CEll *name*

  An optional, repeatable switch and string pair that specifies the name of a library cell. Use this option to display all faults associated with a specified library cell. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -MOdule *name*

  An optional switch and string pair that specifies the name of a module. Use this option to display all faults associated with a specified module. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -INSTance *name*

  An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -SOurce

  An optional switch that outputs the UDFM file name and line number where the fault is defined.

Power-Aware Options (applicable only after you have loaded CPF/UPF power data)

- -ON_domains

  An optional switch that reports faults on all power-on domains.

- -OFF_domains

  An optional switch that reports faults on all power-off domains.

- -Power_domains {*domain_name* …}

  An optional switch and repeatable string that reports faults on the specified power domains (*domain_name*).

- -Isolation_cells

  An optional switch that reports faults on all isolation cells.

- -Level_shifters

  An optional switch that reports faults on all level-shifters.

- -REtention_cells

  An optional switch that reports faults on all retention cells.

Example 1

The following example modifies the current fault population in the tool with the contents of the external fault file, *my_detected_faultlist*, then displays the protected and unprotected faults:

```
set_system_mode analysis
add_faults -all
read_faults my_detected_faultlist -protect
report_faults -all
report_faults -protected
report_faults -unprotected_only
```

Example 2

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y, including the name of the cell associated with the fault:

```
report_faults -stuck_at 0 /mult_81/Y -cell_name

    type   code   pin_pathname  (cell_name)
    ----   ----   ------------------------------------
//    0    DS.CC  /mult_8/Y     (xnor2)
```

Example 3

The following example reports on a fault associated with a primary input:

report_faults -stuck_at 1 /ip_scan_en -cell_name

```
    type   code   pin_pathname   (cell_name)
    ----   ----   ------------------------------------
//    0     AU    /ip_scan_en    primary_input
```

Example 4

The following example shows a fault list with the static test slack and the actual minimum test slack. Only DS faults contain the actual minimum test slack data.

report_faults -delay_data

```
// type    code       pin_pathname
//                                          static_slack   actual_min_slack
// ----    ----      ------------------  -------------  -----------------
    0      DS.CC     /CORE_i/IRQA_B/A
//                                             10.1010       12.3000
    1      DS.CC     /CORE_i/IRQA_B/B
//                                             20.1480       25.4820
    0      AU.BB     /CORE_i/IRQA_B/C
//                                             12.1890        -
    1      AU.BB     /CORE_i/IRQA_B/C
//                                             18.8770        -
```

Example 5

The following example displays information for the stuck-at-0 fault on pin /mult_8/Y plus all isolation cells:

report_faults -stuck_at 0 /mult_81/Y -isolation_cells

Example 6

The following example show the reporting of unlisted faults:

report_faults –unlisted

```
class detections collapsed_fault_count uncollapsed_fault_count
----- ---------- --------------------- -----------------------
UC       0               1252              2079
UC.EAB   0                452               543
DS       1              69873             87232
DS       2              12873             21432
DS       3               9752             11974
DS       4+              4487              6293
AU.BB    0               8708             10046
AU       0               2374              3782
```

Example 7

The following example displays the fault data for *ds* class faults associated with library cells that start with *mux* or *an* to the *my_fault_list* file:

report_faults -class ds -cell mux* -cell an*

```
// Command output

FaultInformation {
   version : 1;
   FaultType (UDFM) {
      FaultList {
         Format : Identifier, Class;
         UdfmType ("intra-cell-bridges_1") {
            Cell ("MUX2") {
               Instance ("/top/i1/i4") {
                  "F2", DS;
               }
            }
            Cell ("AND3") {
               Instance ("/top/i1/i456") {
                  "F2", DS;
                  "F4", DS;
                  "F5", DS;
                  "F7", DS;
               }
            }
         }
      }
   }
}
```

Example 8

The following example shows the default format of the fault report when the fault type is specified as UDFM using the set_fault_type command:

report_faults

```
Class UdfmType    Fault  instance
----- ----------- ----- ----------
UC    my-added-sa f01   /top/i1/i4
DS    my-added_sa f02   /top/i1/i4
DS    intra-cell  f01   /top/i3
AU.BB intra-cell  f02   /top/i3
```

Adding the -source switch to the command extends the report with references to the definition sources:

report_faults -source

```
Class UdfmType    Fault instance   File                        Line
----- ----------- ----- ---------- --------------------------- ----
UC    my-added-sa f01   /top/i1/i4 /home/JohnDoe/myadded.udfm   14
DS    my-added_sa f02   /top/i1/i4 /home/JohnDoe/myadded.udfm  145
DS    intra-cell  f01   /top/i3    /home/JohnDoe/intra.udfm    217
AU.BB intra-cell  f02   /top/i3    /home/JohnDoe/intra.udfm     67
```

Example 9

The following example shows how to report specific AU.TC and AU.PC faults:

report_statistics -detailed_analysis ;# you must first run this command

report_faults fltlist_au.tc -class au.tc 1195 ;# you must supply a gate ID rather than pin pathname

```
type    code    pin_pathname
----   ------   ------------------------------------
  0     AU.TC   /pad_i1/tdo_buf/tri_pin/A
  0     EQ      /pad_i1/tdo_buf/ix68/Y
  1     EQ      /pad_i1/tdo_buf/ix68/A
```

report_faults fltlist_au.pc -class au.pc mode[3] ;# you must supply a pin pathname

```
type    code    pin_pathname
----   ------   ------------------------------------
  0     AU.PC   /mode[3]
  0     EQ      /core_i/test_design_i/pll1/ix17/S0
  0     AU.PC   /core_i/test_design_i/pll1/ix17/A1
  1     AU.PC   /core_i/test_design_i/pll1/ix17/A1
```

Example 10

The following example shows the format of the fault report when the fault type is specified as UDFM and you have enabled multiple detections:

report_faults

```
//  Class  Detections  UdfmType    Fault  Instance
//  -----  ----------  ----------  -----  ---------------
//  DS     2           intra-cell  F1     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F2     ip_inst/pio_i/i_42/inst
//  DS     2           intra-cell  F3     ip_inst/pio_i/i_42/inst
//  DS     3+          intra-cell  F4     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F5     ip_inst/pio_i/i_42/inst
//  DS     1           intra-cell  F6     ip_inst/pio_i/i_42/inst
```

Example 11

The following example shows the format of the fault report when you specify that you want to display faults from a maximum of two levels of hierarchy:

report_faults -hierarchy 2

```
//  /clk_ctrl_clk_testshell                          48
//  /clk_ctrl_clk_testshell/gated_ccb_inst4          6
//  /clk_ctrl_clk_testshell/gated_ccb_inst2          6
//  /clk_ctrl_clk_testshell/i_ipoHold_10275          4
//  /clk_ctrl_clk_testshell/gated_ccb_inst1          10
//  /clk_ctrl_clk_testshell/antiskew_clock_inv       4
//  /clk_ctrl_clk_testshell/antiskew_1               6
//  /clk_ctrl_clk_testshell/gated_ccb_inst3          12
//  /ip_inst                                         240366
//  /ip_inst/dsp_periph_em_ccb_i                     960
//  /ip_inst/siox_i                                  8336
//  /ip_inst/dio_i                                   25218
```

Related Commands

add_faults
analyze_fault
delete_faults
read_faults
read_cpf
read_upf
read_upf
report_power_data
report_testability_data
set_fault_mode
set_fault_protection
set_fault_sampling
set_fault_subclass_analysis
set_fault_type
write_faults

report_feedback_paths

Context: all contexts

Mode: all modes

Prerequisites: You can use this command only after the tool performs the learning process, which happens immediately after flattening a design to the simulation model. Flattening occurs when you first attempt to exit Setup mode or when you issue the create_flat_model command.

Usage

report_feedback_paths [-All | *loop_id#*…][-Display {DEBug | DESign [-Hierarchical] | DAta | ALl} [APpend]] [{> | >>} *file_pathname*]

Description

Displays a report of currently identified feedback paths.

The report_feedback_paths command displays feedback paths the tool identified during the last circuit learning process. By default, the command displays a textual report of all currently identified feedback paths and their identification numbers. Issuing the command with the identification numbers of specific paths of interest will limit the display to just those paths.

You can use the identification numbers with the report_loops -Display command to schematically display specific feedback paths. When you issue this command for specific feedback paths, DFTVisualizer transcripts the same information as the report_feedback_paths command, but only for the specified paths.

Note

Feedback paths include, by default, any duplicated gates.

Arguments

- -ALl

  An optional switch that specifies to report all currently identified feedback paths. This is the default.

- *loop_id#*

  An optional, repeatable, non-negative integer that specifies the identification number of a particular feedback path to report. The tool assigns the numbers consecutively, starting with 0.

- -Display {DEBug | DESign [-Hierarchical] | DAta | ALl} [APpend]

  An optional switch, repeatable literal, and literal that displays the reported information graphically in the specified DFTVisualizer window(s); if *Append* is specified, the information is added to the existing contents of the window instead of replacing the contents.

The choices are as follows:

DEBug — A literal that specifies the Debug window.

DESign — A literal that specifies the Design window. If the *-Hierarchical* literal is specified, only the top-level hierarchical instances are displayed in the Design window; if the *-Hierarchical* literal is not specified, the child instances of the corresponding hierarchical instances are also displayed.

DAta — A literal that specifies the Data window.

ALl — A literal that specifies all the preceding windows.

The default when you specify -Display without including a window name is to add feedback paths to the Debug window.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example enters analysis mode (which flattens the simulation model and performs the learning process), reports the identification numbers of all learned feedback paths, and then displays one of the reported feedback paths (loop #1):

set_system_mode analysis
report_feedback_paths

```
Loop#=0, feedback_buffer=26, #gates_in_network=5
   INV   /I_956__I_582/ (51)
   PBUS  /I_956__I_582/N1/ (96)
   ZVAL  /I_956__I_582/N1/ (101)
   INV   /I_956__I_582/ (106)
   TIEX  /I_956__I_582/ (26)
Loop#=1, feedback_buffer=27, #gates_in_network=5
   INV   /I_962__I_582/ (52)
   PBUS  /I_962__I_582/N1/ (95)
   ZVAL  /I_962__I_582/N1/ (100)
   INV   /I_962__I_582/ (105)
   TIEX  /I_962__I_582/ (27)
```

report_feedback_paths 1 -display design

Related Commands

report_loops -Display

set_loop_handling

report_flattener_rules

Context: all contexts

Mode: all modes

Usage

report_flattener_rules [*rule_id* [{*occurence_id* | -Verbose}]] [{> | >>} *file_pathname*]

Description

Displays either a summary of all the flattening rule violations or the data for a specific violation.

The report_flattener_rules command displays the following information for a specific violation:

- Rule identification number
- Current number of rule failures
- Violation handling

You can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate rules.

Arguments

- *rule_id*

  A literal that specifies the flattening rule violation for which you want to display information. The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

  - Net flattening violations are described in sections “FN1” through “FN9”.
  - Pin flattening violations are described in sections “FP1” through “FP13”.
  - Gate flattening violations are described in sections “FG1” through “FG8”.

- *occurence_id*

  A literal that specifies the identification of the exact flattening rule violation (the occurrence) for which you want to display information. For example, you can analyze the second occurrence of the FG4 rule by specifying the rule_id and the occurence_id, FG4 2. The tool assigns the occurrences of the rules violations as it encounters them; you cannot change either the rule identification number or the ordering of the specific violations.

- -Verbose

  A switch that displays the following for each flattening rule:

  - Rule identification number
  - Number of failures of each rule
  - Current handling status of that rule

  - Brief description of that rule

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example shows the summary information of the FG3 rule:

```
report_flattener_rules fg3
//  FG3: fails=2 handling=warning/noverbose
```

Related Commands

set_flattener_rule_handling

report_gates

Context: all contexts

Mode: setup, analysisUsage

report_gates {***pin_or_net_pathname*** | ***gate_id#*** | ***instance_name***}...
{-Value_trace [position] [-TRANsition] [-Endpoints] [-DISplay]
[-STOP_ambiguous **|** -GO_One_ambiguous **|** -GO_All_ambiguous]}
[-Type {***gate_type*** | **ALLF** | **ALL**}]
[{-Sequential_test_depth | -0Control_depth | -1Control_depth | -Observe_depth}
depth_number]
[-Path {*pin_pathname* | *gate_id*} {*pin_pathname* | *gate_id*}]
[[-Endpoints] [-COnstraints] [-Forward **|** -Backward] {*pin_pathname* | *gate_id*}...]
[{> | >>} *file_pathname*]

Description

Displays the netlist information and simulation results for the specified gates and the simulation results for the specified user-defined ATPG functions.

The report_gates command displays the netlist information for either the design-level or the primitive-level gates you specify. You designate the gate by its gate identification number (ID#), a pathname of a pin connected to a gate, an instance name (design level only), or a gate type. You can specify a design cell by the pathname of a pin on the design cell. If you use a gate ID# or gate type, the command always reports the primitive-level gate. You must flatten the netlist before issuing this command.

The pin_or_net_pathname and instance_name arguments support regular expressions, which may include any number of '*' or '?' wildcard characters embedded in the pathname string. The '*' character matches any sequence of characters (including none) in a name, and the '?' character matches any single character. If a wildcard name is specified, the command will search for matching instance names from the top library cell level down to the primitive gates.

The format for the design-level gate report is:

```
instance_name     cell_type
   input_pin_name   I   (data)   pin_pathname...
...
   output_pin_name   O   (data)   pin_pathname...
...
```

The format for the primitive-level report is:

```
instance_name   (gate_ID#)     gate_type
   input_pin_name   I   (data)   gate_ID#-pin_pathname...
...
   output_pin_name   O   (data)   gate_ID#-pin_pathname...
...
```

When you provide the name or identification number (ID#) of a user-defined ATPG function, the command reports simulation data of the function. The data reported is that specified by the Parallel_pattern option of the set_gate_report command.

The format for the function report is:

```
function_name   (function_ID#)     function_type
   input_pin_name   I   (data)   gate_ID#-pin_pathname...
...
   output_pin_name   O   (data)   gate_ID#-pin_pathname...
...
```

The list associated with the input and output pin names indicate the pins to which they connect. For the primitive level, this also includes the gate index number of the connecting gate and only includes the pin pathname if one exists at that point. There is a limitation on reporting gates at the design level. If some circuitry inside the design cell is completely isolated from other circuitry, the command only reports the circuitry associated with the pin pathname.

You can also report the fan-in or fan-out cone of a specified gate. The endpoints of a cone are defined as the primary inputs, primary outputs, tied gates, rams, roms, flip-flops, and latches. All gates reported are at the primitive level.

You can change the output of the report_gates command by using the set_gate_report command.

Note

The tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, report_gates cannot report simulation values for the EDT logic during these cycles. This is true regardless of the option set with the set_gate_report command.

Backslashes (\), Periods (.) and Spaces in Hierarchical Names

When reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool reads the cell name, “FFH.\E/MY_SCAN_FF .SCAN_FF5” in a WGL pattern, as “FFH/E/MY_SCAN_FF/SCAN_FF5”, and stores the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note

The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

The following example reports on a Verilog gate named “FFH.\E/MY_SCAN_FF .SCAN_FF5”.

report_gates FFH/E/MY_SCAN_FF/SCAN_FF5

```
//  /FFH/E/MY_SCAN_FF/SCAN_FF5 FD1SF
//      D       I  /FFH/E/MY_SCAN_FF/GATE00/Z
//      A       I  /C-4/Z
//      SI      I  /FFG/SO
//      CP      I  /INPBUF1/Z
//      B       I  /C-5/Z
//      Q       O  /GATE20/C
//      QN      O
```

```
//      SO      O  /C-7/A
```

Reporting on the First Input of a Gate

The report_gates command provides a shortcut to display data on the gate connected to the first input of the previously-reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

b

The following example shows how to use report_gates and B commands to trace backward through the first input of the previously reported gate.

report_gates 26

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
b

//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
b

//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
```

If you include an integer argument with “b”, the trace will be backward through the specified input. For example, “b 3” will trace backward through the third input of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

For pins that are not at the library cell boundary (pins internal to the model), the pin name is enclosed in double quotes ("). The following example displays this issue.

set_gate_level primitive

report_gates /I_20/I_226/q

```
//  /I_20/I_226  dffsr
//      clk    I  (HX)  /I_20/I_225/out
//      d      I  (X)   /I_20/I_222/out
//      pre    I  (H1)  /PRE
//      clr    I  (H1)  /CLR
//      q      O  (X)   /I_16/i0  /I_23/I_221/i0  /I_6/i0
//      qb     O  (X)
```

set_gate_level primitive

```
//  Creating schematic for 5 instances (1 was compacted).
```

report_gates /I_20/I_226/q

```
//  /I_20/I_226 (12)  BUF
//      "I0"   I  (0)  39-
//      q      O  (X)  16-/I_16/i0  31-/I_23/I_221/i0
                       17- /I_6/i0
```

b

```
//  /I_20/I_226 (39)  DFF
//      "S"    I  (LX)  26-
//      "R"    I  (LX)  23-
//      clk    I  (HX)  20-/I_20/I_225/out
//      d      I  (X)   36-/I_20/I_222/out
//      "OUT"  O  (0)   12-  13-
//      MASTER  cell_id=1  chain=c1  group=g1 invert_data=FFFF
```

Reporting on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also use a shortcut to trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

f

The following example shows how to use Report Gate and F commands to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

f

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

f

```
//  /u1/inst__565_ff_d_1__13 (268)  LA
//      "S"    I  14-
//      "R"    I  145-
//      BCLK   I  1-/scan_sclk
//      "D0"   I  26-
//      "OUT"  O  24-  25-
```

If you include an integer argument with “f”, the trace will be forward through the specified fanout of the previously reported gate. For example, “f 2” will trace forward through the second fanout of the previously specified gate. When using integer arguments with the B and F commands in the tool, all arguments must be given at the primitive level.

Note

When reporting gates, be aware that a reported X may not have been captured. For example, when the tool uses scan chain masking, it records the actual measured value for each cell only in the unmasked, selected scan chain in a compactor group; the rest of the scan chains in the group are masked, so the tool records all Xs for their scan cells. When not using scan chain masking, the tool records an X for a scan cell if it is made unmeasurable as a result of the actual occurrence of an X in the corresponding cell of a different scan chain in the same compactor group. For more information, see "Understanding Scan Chain Masking in the Compactor" in the *Tessent TestKompress User's Manual*.

When using the tool to report on RAM or ROM gates, the report_gates command displays the RAM and ROM data that describes their behavior. You can use the set_gate_report command to customize the content of the reported data. For example, assume you add_faults on a RAM pin, then generate RAM sequential patterns. If you follow that with a set_gate_report command, specifying the Parallel_pattern argument, the report_gates output shows for the specified pattern, the address and data written to the RAM. A hyphen (-) separates the information for each cycle. The following example shows how this information is displayed for an 8-bit output named "OUT" and a 4-cycle pattern.

```
"OUT" O (S[0]=00000011-S[0]=00001001-S-SX) 5353- 5354- 5355-
                                           5356- 5357- 5358-
                                           5359- 5360-
```

In the first cycle, the pattern writes 00000011 to address 0, in the second, it writes 00001001 to address 0. The pattern does not write anything in the third cycle. The fourth cycle is a "don't care" cycle.

The RAM and ROM simulation primitives are the same as the library primitives with the outputs being OUT gates in the RAM/ROM fanout list. The report_gates command gives RAM behavior summary information at the end of the displayed data. The report displays messages similar to the following:

```
write port: write=G/G (vvv-v-v) first_adr=G   first_di=Gstability=vv
read port: read=G/G (vvv-v-v)   first_adr=G   first_do=Gstability=vv
Test behavior: stability=vvvv   tiex-X=v      read_only=v
                          ram_sequential=v/v seq_transparent=v/v
Contention Behavior: write_write=v/v   read_read=vread_write=v/v
```

A RAM is classified as a read-only RAM if all three of the following conditions are met:

- its contents cannot be disturbed by any procedure except test setup;
- its set, reset and write ports must hold at an inactive state in the capture window;
- it has an initialization file defined in the ATPG library.

The following describes the fields for the write port message line:

Identifies the gate index number of the first data input line for a given write port. In the list of gate inputs, the other data inputs will immediately follow.

Identifies the gate index number of the first address line for a given write port. In the list of gate inputs, the other address inputs will immediately follow.

```
write port:   write=G/G (vvv-v-v)   first_adr=G   first_di=G   stability=vv
```

Identifies the behavior of the write clock and write enable lines for a given write port:

Identifies the following stability behavior for a given write port, respectively:
- write control stability
- write load stability

The values can be either of the following:
- S (stable)
- U (unstable)

G/G — Two gate index numbers separated by a “/”.
- First G is the write clock line.
- Second G is the write enable line. A “-” means there is no enable line.

(vvv-v-v) — Up to five values indicating the behavior of the following conditions, respectively:
- clock_off_enable_off
- clock_on_enable_off (Not shown if there is no enable line.)
- clock_off_enable_on (Not shown if there is no enable line.)
- two optional clock information values separated by dashes (-)

The first three values can be any of the following:
- X (write X)
- 0 (write 0)
- 1 (write 1)
- H (hold old values)
- P (potential write)

The last two values can be either of the following:
- CLK (write clock connected to scan clock)
- IH (write clock line is inactive high)

The following describes the fields for the read port message line:

Identifies the gate index number of the first address line for a given read port. In the list of gate inputs, the other address inputs will immediately follow.

Identifies the gate index number of the first data output line for a given read port. In the list of gate outputs, the other data outputs will immediately follow.

When reporting primitives, the output of a primitive is listed last and marked with an “O”. For ROMs/RAMs with multiple outputs, the “O” is followed by a list of OUT gates, each representing one data output bit of the memory. First_do will appear in this list followed by, in order, any remaining data output bits from the same read port.

```
read port:   read=G/G (vvv-v-v)   first_adr=G   first_do=G   stability=vv
```

Identifies the following stability behavior for a given read port, respectively:
- read control stability
- read load stability

The values can be either of the following:
- S (stable)
- U (unstable)

Identifies the behavior of the read clock and read enable lines for a given write port:

G/G — Two gate index numbers separated by a “/”.
- First G is the read clock line.
- Second G is the read enable line. A “-” means there is no enable line.

(vvv-v-v) — Up to five values indicating the behavior of the following conditions, respectively:
- clock_off_enable_off
- clock_on_enable_off (Not shown if there is no enable line.)
- clock_off_enable_on (Not shown if there is no enable line.)
- two optional clock information values separated by dashes (-)

The first three values can be any of the following:
- X (read X)
- 0 (read 0)
- 1 (read 1)
- H (hold old values)
- P (potential read)

The last two values can be either of the following:
- CLK (read clock connected to scan clock)
- IH (read clock line is inactive high)

The following describes the fields for the Test behavior message line:

The following describes the fields for the Contention Behavior message line:

Arguments

The following four arguments list the four methods for naming the objects on which you want the tool to display information. You can use any number of the four argument choices, in any order.

- ***function_name***

  A repeatable string that specifies a user-defined ATPG function.

- ***gate_id#***

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

- ***pin_or_net_pathname***

  A repeatable string that specifies the pathnames of pins or nets in the design netlist. You may use wildcard characters to match multiple pin or net pathnames.

  For a hierarchical pathname, the display will include information describing how that pathname maps to the driving design level pin(s) and gate(s) for which data is displayed.

- ***instance_name***

  A repeatable string that specifies the hierarchical pathname of an instance of a library cell within the design. If a valid library instance pathname is given when in primitive level, all pins on that library cell are reported. When in primitive level, *instance_name* may also be the pathname of a primitive instance.

- -Value_trace [*position*] [-TRANsition] [-Endpoints] [-DISplay] [-STOP_ambiguous | -GO_One_ambiguous | -GO_All_ambiguous]

  An optional switch that specifies to trace backward from the output pin of the gate to the source of a value. The -Value _trace option only supports the primitive report level; therefore, all arguments must be specified at the primitive level.

  position

  An optional value that specifies which bit to trace on for multi-bit data. For example, if a clock signal is 0X0 for a shift cycle, you can specify the tracing is on the second bit (X). The default is to use position 0 (the first bit).

  -TRANsition

  An optional switch that specifies to trace backward to the source of the transition of a pair of values. When using the *position* option, *position* specifies the first bit of the transition.

  -Endpoints

  An optional switch that specifies to report only endpoints that control the specified gate. The result of using this option also depends on the tracing mode. Only when using the -GO_All_ambiguous tracing mode are all endpoints reported.

  -DISplay

  An optional switch that displays all tracing gates in the Debug window of DFTVisualizer.

Note

The -Display switch cannot be used in conjunction with the -Path switch.

The following three optional switches let you choose the tracing mode of operation.

-STOP_ambiguous

An optional switch that specifies to stop if either input could cause the value on the output. This is the default setting if -go_one_ambiguous or -go_all_ambiguous is not specified.

-GO_One_ambiguous

An optional switch that specifies to trace one input if multiple input could cause the output value. For example, if the output of an OR is X, input A is X, input B is X, then the source of X is represented by both A and B. The tool will pick one controlling input, say B, and continue tracing input B.

then the source of X is represented by B. The tool will continue tracing input B.

-GO_All_ambiguous

An optional switch that specifies to trace all inputs if multiple input could cause the output value.

- -Type *gate_type*

  An optional repeatable switch where *gate_type* is a name pair that specifies the gate types for which you want to display the gate information. Table 7-28 and lists the supported types.

  The **ALLF** option lets you report gates on all primitive level gates and ATPG functions. This feature supports users who require access to the tool's flattened models. The **ALL** option lets you report gates on all primitive level gates (but not functions).

- -Forward {*pin_pathname* | *gate_id*}...

  An optional switch that reports the fan-out cone of the specified gate.

- -Backward {*pin_pathname* | *gate_id*}...

  An optional switch that reports the fan-in cone of the specified gate.

- -Endpoints [-Forward | -Backward] {*pin_pathname* | *gate_id*}...

  An optional switch that reports only the endpoint of the cone.

Note

Immediately after -Endpoints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

- -COnstraints [-Forward | -Backward] {*pin_pathname* | *gate_id*}...

  An optional switch that takes into account the effects of pin and cell constraints.

  For a gate whose output is constrained to a fixed value, the tools report only other gates whose output is also constrained. For a gate whose output is not constrained to a fixed value, the tools report only other gates whose outputs are not constrained. During backwards tracing, a mux input, which is always blocked by a constrained value on the select line of the mux, will never be backtraced.

Note

Immediately after -Constraints, you must specify either -Forward or -Backward followed by the specified gate. When using -Endpoints or -COnstraints simultaneously, -Forward or -Backward followed by the specified gate need only be entered once.

- -Path {*gate_id#* | *pin_pathname*} {*gate_id#* | *pin_pathname*}

Note

This switch cannot be used in conjunction with the -*Display* switch.

An optional switch that reports on the path between the first *gate_id#* or *pin_pathname* and the second *gate_id#* or *pin_pathname*. All paths must be specified at the primitive level. Paths do not extend through sequential elements. The output generates a report on each primitive in the path(s), in order of increasing gate_id#.

- -Sequential_test_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a sequential test depth greater than the specified *depth_number*. The sequential test depth is the greater of the control 0 and control 1 depths plus the observe depth for the primitive output. This is the minimum depth required to test stuck faults at the primitive output. This is also the number used to determine the maximum sequential test depth of the circuit.

- -0Control_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a control 0 depth greater than the specified *depth_number*. The control 0 depth is the minimum sequential depth required to set the primitive output to a 0.

- -1Control_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with a control 1 depth greater than the specified *depth_number*. The control 1 depth is the minimum sequential depth required to set the primitive output to a 1.

- -Observe_depth *depth_number*

  An optional switch and integer pair that specifies for the tool to report gates with an observe depth greater than the specified *depth_number*. The observe depth is the minimum sequential depth required to observe the output of the primitive at either a primary output or scan cell.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair used, at the end of the argument list, for appending to the contents of *file_pathname*.

Table 7-28. Reportable Gate Types

gate_type	Description
BUF	buffer
INV	inverter
AND	and
NAND	inverted and
OR	or
NOR	inverted or
XOR	exclusive-or
NXOR	inverted exclusive-or
DFF	D flip-flop, same as _dff library primitive
LA	latch, same as _dlat library primitive
PI	primary input
PO	primary output
TIE0	tied low
TIE1	tied high
TIEX	tied unknown
TIEZ	tied high impedence
HIST	histogram of each primitive type used
TSD	tri-state driver, first input is active high enable line
BUS	tri-state bus
ZVAL	Z converter gate, converts Z to X
WIRE	undetermined wired gate
MUX	2-way multiplexor, first line is select line
SW	switch gate, first input is active high enable line
PBUS	pulled bus gate, where the second input is the pulled value
OUT	memory model gate, created for each read data bit
RAM	random access memory
ROM	read only memory

Table 7-28. Reportable Gate Types (cont.)

gate_type	Description
XDET	X detector, gives 1 when input is X
ZDET	Z detector, gives 1 when input is Z
TLA	transparent latch
STLA	seq_transparent latch
STFF	seq_transparent flip-flip
FB_BUF	internal gate type to break combinational loops in the design

Table 7-29. Tessent FastScan and Tessent TestKompress Clock Port Categories

Category	Description
IL	inactive low
IH	inactive high
AHS	active high slave
ALS	active low slave

For more information on using the clock port options, refer to "The ATPG Analysis Option."

Example 1

The following example displays the simulated values of the gate and its inputs at the primitive level:

```
set_system_mode analysis
set_gate_report error_pattern
set_gate_level primitive
report_gates i_1006/o
```

The gate report for the design level may look like the following:

```
/P2.13P  ND2
    A          I  /LD.1
    B          I  /M1.1
    Z          O  /P2.2P/S
```

The gate report for the primitive level may look like the following:

```
/P2.13P (23)  NAND
    A          I  9-/LD.1
    B          I  6-/M1.1
    Z          O  33-/P2.2P/S
```

The gate report for the primitive level of a RAM gate may look like the following:

```
//  /P1.RAM/U1 (67)  RAM
//      "I0"   I  27-
//      "I1"   I  20-
//      RCK0   I  36-
//      "I3"   I  26-
//      "I4"   I  42-
//      "I5"   I  43-
//      "I6"   I  44-
//      "I7"   I  45-
//      "I8"   I  46-
//      WCK0   I  28-
//      "I10"  I  19-
//      A14    I  29-
//      A13    I  30-
//      A12    I  31-
//      A11    I  32-
//      A10    I  34-
//      D14    I  66-/P1.RAM/D1[0]
//      D13    I  65-/P1.RAM/D1[1]
//      D12    I  64-/P1.RAM/D1[2]
//      D11    I  63-/P1.RAM/D1[3]
//      D10    I  62-/P1.RAM/D1[4]
//      "OUT"  O  68-  69-  70-
//                71-  72-
//      address size =        5
//      data size =           5
//      minimum address =     0
//      maximum address =     31
//      number write ports =  1
//      number read ports =   1
//      edge_triggered =      no
//      initialization file = ram.init_file
//      write port: write=28/- (H)  first_adr=29  first_di=66
                                stability=SS
//      read port:  read=36/- (0)  first_adr=42  first_do=68
                                stability=SS
//      Test behavior: stability=SSSS  tie-X=0  read_only=1
                                ram_sequential=1/0  seq_transparent=0/0
//      Contention behavior: write_write=allow/X  read_read=normal
                                read_write=read_new/write_normal
```

Example 2

The following example defines a function named “my_function” and displays all its simulation results from the last simulation pass:

set_system_mode analysis
add_atpg_functions my_function or /g1 ~/g2
set_gate_report parallel_pattern all
report_gates my_function

```
//  my_function (44)  USER_OR
//      "I0"    (0/0/0/0/0/0/0/0.../0) 12-/g1
//      ~"I1"   (1/1/1/1/1/1/1/1.../1) 13-/g2
//      "OUT"   (0/0/0/0/0/0/0/0.../0)
```

Example 3

The following example shows the use of wildcards:

report_gates /xscan_0_0_cch_scan_32x/ix15*

```
// /xscan_0_0_cch_scan_32x/ix151 NOR2XL
// A  I  /myop1<0> [1]
// B  I  /myop2[1]
// Y  O  /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A0
// /xscan_0_0_cch_scan_32x/ix153 NAND2X1
// A  I  /myop1<0> [0]
// B  I  /myop2[0]
// Y  O  /xscan_0_0_cch_scan_32x/sum_add_0_ix1/A \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix23/B \
         /xscan_0_0_cch_scan_32x/sum_add_0_ix27/A1
// /xscan_0_0_cch_scan_32x/ix155 NAND2X1
// A  I  /myop1<0> [1]
// B  I  /myop2[1]
// Y  O  /xscan_0_0_cch_scan_32x/ix166/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix27/B0
// /xscan_0_0_cch_scan_32x/ix157 NAND2X1
// A  I  /myop1<0> [3]
// B  I  /myop2[3]
// Y  O  /xscan_0_0_cch_scan_32x/ix174/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```

report_gates /xscan_0_0_cch_scan_32x/ix159

```
// /xscan_0_0_cch_scan_32x/ix159 NAND2X1
// A  I  /myop1<0> [5]
// B  I  /myop2[5]
// Y  O  /xscan_0_0_cch_scan_32x/ix188/B0 \
         /xscan_0_0_cch_scan_32x/ sum_add_0_ix83/B0
```

set_gate_level primitive

report_gates /xscan_0_0_cch_scan_32x/ix157

```
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
// /xscan_0_0_cch_scan_32x/ix157 (43) NAND
// A  I  7-/myop1<0> [3]
// B  I  17-/myop2[3]
// Y  O  60-/xscan_0_0_cch_scan_32x/ix174/B0 \
         75-/xscan_0_0_cch_scan_32x/ sum_add_0_ix55/B0
```

Example 4

The following example shows how the output report will change if the input pathname is a hierarchical pin or net. In this case, an additional line is output at the top of the report, indicating the mapping that was found.

set_gate_level design

report_gates /sub3/in2

```
// Hierarchical pin /sub3/in2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//    A1 I /in1
//    A0 I /sub1/gate2/Y
//    Y  O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//       /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

report_gates /w2

```
// Hierarchical net /w2 maps to /sub1/gate4/Y
// /sub1/gate4 nand02
//    A1 I /in1
//    A0 I /sub1/gate2/Y
//    Y  O /sub3/gate1/A1 /sub3/micro1/gate1/A1 /sub2/gate3/A1
//         /sub3/gate3/A1 /sub3/micro1/gate2/A1 /sub2/gate1/A1
```

Example 5

The following example traces backward to the source of a DRC error:

```
//  Error: clock input of /ff2 (35) set to X. (T5-1)
//  Error: Scan chain c0 blocked at gate /ff2 (35) after tracing 1 cells.
(T3-1)
//  Error: Rules checking unsuccessful, cannot exit SETUP mode.
```

set_gate_level prim

set_gate_report drc shift

report_gates 35

```
//  /ff2 (35)  DFF
//      "S"    I  (000)  10-
//      "R"    I  (000)  9-
//      CLK    I  (0X0)  27-/gate8/Y
//      "D0"   I  (XXX)  32-
//      "OUT"  O  (XXX)  14-  15-
```

report_gates 27 –v 1

```
//  /gate8 (27)  AND
//      A0     I  (010)  6-/clk1
//      A1     I  (XXX)  21-/ff4/Q
//      Y      O  (0X0)  35-/ff2/CLK
//
//  /ff4 (21)  INV
//      "I0"   I  (XXX)  11-
//      Q      O  (XXX)  27-/gate8/A1
//
//  /ff4 (11)  INV
```

```
//      "I0"   I  (XXX)  33-
//      "OUT"  O  (XXX)  21-  22-
//
//  /ff4 (33)  DFF
//      "S"    I  (000)  10-
//      "R"    I  (000)  9-
//      CLK    I  (010)  7-/clk2
//      D      I  (XXX)  3-/in3
//      "OUT"  O  (XXX)  11-
//
// Total 4 gates traced which controls gate /gate8 (27) to X (bit 1).
```

Example 6

The following example traces backward to the first ambiguous gate where multiple source control the values:

set_gate_level prim

set_gate_report drc shift

report_gates 29

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
```

report_gates29 –v 1

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//      A0     I  (XXX)  21-/ff0/Q
//      A1     I  (XXX)  16-/ff2/QB
//      Y      O  (XXX)  29-/sgate1/A0
// Total 2 gates traced which controls gate /sgate1 (29) to X (bit 1).
```

Example 7

The following example traces backward to all ambiguous gates:

report_gates 29 –v 1 –go_all

```
//  /sgate1 (29)  AND
//      A0     I  (XXX)  25-/sgate2/Y
//      A1     I  (111)  10-
//      Y      O  (XXX)  31-/ff1/SI
//
//  /sgate2 (25)  OR
//      A0     I  (XXX)  21-/ff0/Q
//      A1     I  (XXX)  16-/ff2/QB
//      Y      O  (XXX)  29-/sgate1/A0
//
//  /ff0 (21)  INV
//      "I0"   I  (XXX)  12-
//      Q      O  (XXX)  25-/sgate2/A0
```

```
//
//  /ff0 (12)  INV
//      "I0"   I  (XXX)  35-
//      "OUT"  O  (XXX)  21-  22-
//
//  /ff0 (35)  DFF
//      "S"    I  (000)  11-
//      "R"    I  (000)  9-
//      CLK    I  (010)  6-/clk1
//      D      I  (XXX)  3-/in3
//      "OUT"  O  (XXX)  12-
//
//  /ff2 (16)  INV
//      "I0"   I  (XXX)  37-
//      QB     O  (XXX)  25-/sgate2/A1
//
//  /ff2 (37)  DFF
//      "S"    I  (000)  11-
//      "R"    I  (000)  9-
//      CLK    I  (010)  6-/clk1
//      "D0"   I  (XXX)  34-
//      "OUT"  O  (XXX)  15-  16-
// Total 7 gates traced which controls gate /sgate1 (29) to X (bit 1).
```

Example 8

The following example traces backward from the output of gate 23541 to the origin of the transition:

```
//  Fault type is transition
```

set_gate_level prim
set_gate_report pat3
report_gates 23541 -v -trans

```
//  /p2/pic1/alu/U393 (23541)  NAND
//      A2     I  (1-1)  23366-/p2/pic1/alu/U377/Y
//      A1     I  (1-X)  22150-/p2/pic1/alu/U344/Y
//      A3     I  (1-1)  23178-/p2/pic1/alu/U187/Y
//      A0     I  (1-1)  20580-/p2/pic1/alu/U366/Y
//      Y      O  (0-X)  23742-/U133/A  23741-/p2/pic1/U1185/A  24371-
/p2/pic1/regs/picdram/din<3>
//                       23740-/p2/pic1/alu/U310/A
//  /p2/pic1/alu/U344 (22150)  BUF
//      A      I  (1-X)  21714-/p2/pic1/alu/U448/Y
//      Y      O  (1-X)  23541-/p2/pic1/alu/U393/A1
//  /p2/pic1/alu/U448 (21714)  INV
//      "I0"   I  (0-X)  21247-
//      Y      O  (1-X)  22150-/p2/pic1/alu/U344/A
//  /p2/pic1/alu/U448 (21247)  MUX
//      S0     I  (0-X)  18063-/p2/pic1/alu/U374/Y
//      A0     I  (0-X)  18072-/p2/pic1/alu/U423/Y
//      A1     I  (0-1)  20665-/p2/pic1/alu/U422/Y
//      "OUT"  O  (0-X)  21714-
//
//  Total 4 gates traced which control gate /p2/pic1/alu/U393/Y (23541) to
L_X transition (bit 0-1).
```

Example 9

The following example displays the status of the faults on the gate and its inputs:

set_gate_report fault_status
report_gates /ram1/address<4>

```
//  /ram1  picdram
//      clk    I  (AU.SEQ:AU.SEQ)  /gater2/an_4/Y
//      address<6>  I  (AU.SEQ:AU.SEQ)  /address1[6]
//      address<5>  I  (AU.SEQ:AU.SEQ)  /address1[5]
//      address<4>  I  (AU.SEQ:AU.SEQ)  /address1[4]
//      address<3>  I  (AU.SEQ:AU.SEQ)  /address1[3]
//      address<2>  I  (AU.SEQ:AU.SEQ)  /address1[2]
```

report_gates /an_10/A1

```
//  /an_10  and03
//      A0     I  (AU.BB:AU.BB)  /sf_1_6/Q
//      A1     I  (AU.BB:AU.BB)  /sf_3_6/Q
//      A2     I  (AU.BB:AU.BB)  /sf_5_6/Q
//      Y      O  (AU.BB:AU.BB)  /po6
```

report_gates /gater11/an_1/Y

```
//  /gater11/an_1  inv02
//      A      I  (AU.TC:AU.TC)  /an_13/Y
//      Y      O  (AU.TC:AU.TC)  /gater11/lat1/CLK
```

In the example, the tool reports the following untestable (AU) fault sub-classes:

- BLACK_BOXES (BB) — Untestable faults due to black boxes.
- SEQUENTIAL_DEPTH (SEQ) — Untestable faults with the current sequential_depth and may be testable with a higher depth.
- TIED_CELLS (TC) — Untestable faults due to tied or blocked tied non-scan cells.

For a complete list of these AU fault sub-classes, see Table 7-49.

Example 10

The following example shows how a launch-off-shift transition pattern can be created and reported:

set_fault_type transition
create_patterns
set_gate_report pattern 0
report_gates /pad_ring/padi/muxpadi_pk0/g1514/x

```
//  /pad_ring/padi/muxpadi_pk0/g1514  exor3_lolkhvt_2
//      a      I  ((0)-00)  /pad_ring/padi/muxpadi_pk0/g1515/x
//      b      I  ((1)-00)  /pad_ring/padi/muxpadi_pk0/g1525/x
//      c      I  ((1)-10)  /pad_ring/padi/muxpadi_pk0/g1518/x
//      x      O  ((0)-10)  /pad_ring/padi/muxpadi_pk0/g1513/d1
```

In the example, the first value in parentheses is the value from the last scan-shift event. The next values (following the dash “-”) are the first capture-cycle values. In this case there are two events for the first capture cycle.

Example 11

The following example shows how the value used to simulate a gate is displayed (in **bold**) when that gate is reported as not simulated:

```
//  command: rep gate /piccpu_i/status_reg_7/DFF1/D    // not simulated
//  /piccpu_i/status_reg_7/DFF1 (2520)  DFF
//      "S"    I  66-
//      "R"    I  67-
//      CP    I  543-
//      D      I  2301-/piccpu_i/status_reg_7/MUX2/Q
//      "OUT"  O  235-
//      MASTER  cell_id=0  chain=chain7  group=grp1  invert_data=FFFF
//
//  Proc: shi 1 sh 2 sh 3 sh 4 sh 5 sh 6 sh 7 sh 8 sh 9 sh10 sh11 cap
//       ----- ----- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
//        i
//        n 123  123  123  123  123  123  123  123  123  123  123 o o
//  Time: i 000  000  000  000  000  000  000  000  000  000  000 f f
//        t0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 fXf
//       ----- ----- ---- ---- ---- ---- ---- ---- ---- ---- ---- ----
//  Sim:  XXXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX XX11 1XX
//
//  Inputs:
//  "S"   00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 000
//  "R"   00000 0000 0000 0000 0000 0000 0000 0000 0000 0000 0000 000
//  CP    00010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0010 0X0
//  D     was not simulated during shift procedure in DRC K19. Simulation
  value used = X.
```

Related Commands

set_gate_level

set_gate_report

report_graybox_statistics

Context: patterns -scan

Mode: analysis

Usage

report_graybox_statistics { [-top *integer*] | [-percentage_exceeds *integer*] | [-exceeds_max_sequential_levels] }

Description

Reports the statistics gathered by graybox analysis.

This command works only after running graybox analysis using the -collect_reporting_data switch of the analyze_graybox command.

Arguments

- -top *integer*

  An optional switch and integer pair that lists the graybox analysis backward tracing origins with the highest number of leaf instances. The *integer* argument specifies how many tracing origins from the top will be listed in the report output. A tracing origin can be a PO, a wrapper cell input pin, or a preserve instance.

- -percentage_exceeds *integer*

  An optional switch that lists all tracing origins that exceed the specified percentage of leaf instances (with respect to all leaf instances in design) covered during backtracing.

- -exceeds_max_sequential_levels

  An optional switch that lists the tracing origins that exceed the maximum sequential levels limit specified by “analyze_graybox -max_sequential_levels”. The default value for maximum sequential levels is 2 unless specified by the -max_sequential_levels switch.

Example

The following example shows the command output for the top four tracing origins with the highest number of leaf instances:

report_graybox_statistics -top 4

```
-------------------------------------------------------------------------------
                                Number of       Percentage   Sequential Levels
Start Point  Pin Type           Leaf Instances  of Design    Limit Reached
-----------  ------------------ --------------  -----------  -----------------
sff2/D,SI    WrapperCellDataPin  3               50.00        yes
sff1/D       WrapperCellDataPin  2               33.33        -
po           PrimaryOutput       1               16.67        -
so           PrimaryOutput       1               16.67        -
-------------------------------------------------------------------------------
```

The explanation of the columns are as follows:

- Start Point is the pathname of the tracing origin.

- Pin Type is one of the following: PrimaryOutput, WrapperCellDataPin, WrapperCellSetPin, WrapperCellResetPin, WrapperCellClockPin, PreserveInstanceInputPin.
- Number of Leaf Instances is the number of leaf instances marked for the tracing origin.
- Percentage of Design is the percentage of covered leaf instances with respect to the leaf instances in the entire design.
- Sequential Levels Limit Reached is “yes” when the specified or default maximum sequential levels limit is exceeded, and is “-” otherwise.

Related Commands

analyze_graybox

report_icl_extraction_options

Context: all contexts

Mode: all modes

Usage

report_icl_extraction_options

Description

Provides a human readable report of the ICL extraction options.

Arguments

- None

Example

The following command displays the ICL extraction settings in a human readable form:

report_icl_extraction_options

```
//  Option                | Type | Value
//  ----------------------+------+------
//  extract_learned_muxes | BOOL |    on
//  extract_library_muxes | BOOL |    on
```

Related Commands

get_icl_extraction_options
set_icl_extraction_options

report_icl_modules

Context: dft, patterns

Mode: all modes

Usage

report_icl_modules [-all | -modules <modules> [-hierarchical]]

Description

Reports loaded and extracted ICL modules in either ICL syntax or human readable form. The output can be redirected to a file.

If you invoke this command without any arguments, the names of all loaded and extracted ICL modules are reported in human readable form.

Note

If you invoke this command during setup mode, before ICL extraction has taken place, only the loaded ICL modules will be reported. If you invoke this command in the analysis mode, the tool may have performed ICL extraction, and the report_icl_modules command will report the loaded and extracted ICL modules.

Arguments

- -all

  An optional switch that reports all loaded and extracted ICL modules in ICL syntax.

- -modules <modules>

  An optional switch that reports the specified modules in ICL syntax. The switch can take an object_spec (TCL string list or object list).

- -hierarchical

  An optional switch that reports the listed modules, as well as a recursive listing of all modules that are instantiated by the listed modules and their child modules.

Example 1

In the following example, when you issue this command without any arguments, it reports the names of all the ICL modules in human readable form:

report_icl_modules

```
The following ICL modules have been loaded:
                  chip, tap, sib, tap1_fsm
```

Example 2

The following example reports the ICL syntax for the module chip:

report_icl_modules -modules chip

```
Module chip {
   // ICL module read from source on or near line 1 of file
'./data/icl/chip3.icl'
   TCKPort tck;
   ScanInPort tdi;
   ScanOutPort tdo {
      Source MyTap.tdo;
   }
   TMSPort tms;
   TRSTPort trst;
   Instance MyTap Of tap {
      InputPort tck = tck;
      InputPort tdi = tdi;
      InputPort tms = tms;
      InputPort trst = trst;
      InputPort fromTdr1 = MySib2.so;
   }
}
```

Example 3

The following example reports the ICL syntax for the module chip as well as a recursive listing of all modules instantiated by chip and its child modules.

report_icl_modules -modules chip -hierarchical

```
Module chip {
   // ICL module read from source on or near line 1 of file
'./data/icl/chip.icl'
   TCKPort tck;
   ScanInPort tdi;
   ScanOutPort tdo {
      Source MyTap.tdo;
   }
   TMSPort tms;
   TRSTPort trst;
   Instance MyTap Of tap {
      InputPort tck = tck;
      InputPort tdi = tdi;
      InputPort tms = tms;
      InputPort trst = trst;
      InputPort fromTdr1 = MySib2.so;
   }
   Instance MySib1 Of sib {
      InputPort si = tdi;
      InputPort se = MyTap.se;
      InputPort ce = MyTap.ce;
      InputPort ue = MyTap.ue;
      InputPort en = MyTap.tdrEn1;
      InputPort tck = tck;
      InputPort fso = MyTdr1.so;
```

```
      }
   }

   // instanced as chip.MyTap
   Module tap {
      // ICL module read from source on or near line 1 of file
   './data/icl/tap.icl'
      TCKPort tck;
      ScanInPort tdi;
      ScanOutPort tdo {
         Source IRMux;
      }
      TMSPort tms;
      TRSTPort trst;
      ToSelectPort tdrEn1 {
         Source sel1;
      }
      ScanInPort fromTdr1;
      ToSelectPort tapEn1 {
         Source IR[2];
      }
      ToCaptureEnPort ce;
      ToShiftEnPort se;
      ToUpdateEnPort ue;
      ScanInterface TAP {
         Port tdi;
         Port tdo;
         Port tms;
      }
      ScanInterface Internal {
         Port fromTdr1;
         Port tdo;
         Port tdrEn1;
         Port ce;
         Port se;
         Port ue;
      }
      ScanRegister IR[2:0] {
         ScanInSource tdi;
         CaptureSource 3'b001;
         ResetValue 3'b000;
      }
      ScanRegister bypass {
         ScanInSource tdi;
         CaptureSource 1'b0;
      }
      ScanMux IRMux SelectedBy fsm.irSel {
         1'b0 : DRMux;
         1'b1 : IR[0];
      }
      ScanMux DRMux SelectedBy IR[2:0] {
         3'bx00 : bypass;
         3'bx01 : fromTdr1;
      }
      LogicSignal sel1 {
         IR[2:0] == 3'bx01;
      }
      Instance fsm Of tap1_fsm {
```

```
        InputPort tck = tck;
        InputPort tms = tms;
        InputPort trst = trst;
     }
  }

  // instanced as tap.fsm
  Module tap1_fsm {
     // ICL module read from source on or near line 59 of file
  './data/icl/tap.icl'
     TCKPort tck;
     TMSPort tms;
     TRSTPort trst;
     ToIRSelectPort irSel;
     ToResetPort tlr;
  }
```

Related Commands

get_icl_instances
get_icl_modules
read_icl

report_iclock

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

Mode: analysis

Prerequisites: In the patterns -ijtag context, a pattern set has been opened by the open_pattern_set command.

Usage

report_iclock [-overridden]

Description

Reports the ICL ClockPort specified by the iClock command as well as their extracted sources and cumulative freqMultiplier and freqDivider values.

Arguments

- -overridden

  Specifies to return the ICL port and pin names on which an iClockOverride command was issued as opposed to the iClock command when the option is not specified.

Return Value

None

Example

The following example shows the result of the report_iclock command specified inside a pattern set in which two iClock commands have been issued. The first iClock command is on the reference pin of a PLL. The root-level clock port called CLK is its source and the connection is direct. The second one is on the CLK input of a memory BIST instance.

The report reflects the fact that the source tracing went through the PLL and picked up a FreqMultiplier value of 10 and a FreqDivider value of 2 either from the ICL definition or from a iClockOverride command that could have been specified on the output clock port of the PLL as shown in the example provided for the iClock command.

report_iclock

```
// iClock     Source  FreqMultiplier  FreqDivider
// =======    =====   ==============  ===========
// PLL.ref    CLK                 1            1
// mbist.clk  CLK                 10           2
```

Related Commands

add_clocks
delete_clocks
get_iclock_list
get_iclock_option
iClock
iClockOverride
iRunLoop
report_clocks

report_id_stamp

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_id_stamp [{> | >>} *file_pathname*]

Description

Displays the unique identifier that the tool assigns each internal pattern set.

The report_id_stamp command displays the current internal pattern set's unique identification stamp which consists of five fields, each separated by a colon (:). Due to the length of three of the fields, the tool encodes those fields and displays the encoded information. The tool encodes the three fields using 4 bytes of hexadecimal numbers. This encoding's primary use is to ensure that each pattern set has a unique identification stamp. The following list shows the information each field provides:

1. Tool version number
2. Encoded environment settings
3. Encoded DRC rules data
4. Number of patterns in the internal pattern set
5. Encoded pattern data

The tool generates the identification stamp each time you issue either the report_id_stamp command or the write_patterns -Environment command. You can use the identification stamp to tag identical patterns saved in different formats.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example displays the unique identification stamp for the current pattern set:

report_id_stamp

```
2012.3:5c95:3e10:16:1bf2
```

Related Commands

write_patterns

report_iddq_exceptions

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

Mode: setup, analysis

Usage

report_iddq_exceptions [{> | >>} *file_pathname*]

Description

Displays current exceptions to Iddq restrictions.

The report_iddq_exceptions command displays Iddq exceptions added with the add_iddq_exceptions command.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example displays Iddq exceptions on sites for which the tool would report Iddq check violations if the exceptions did not exist.

```
report_iddq_exceptions
```

Related Information

For more information on IDDQ, refer to "Creating an IDDQ Test Set" in the *Scan and ATPG User's Manual*.

Related Commands

add_iddq_exceptions
analyze_restrictions
delete_iddq_exceptions
report_gates
report_primary_outputs
set_iddq_checks

report_ijtag_instances

Context: dft patterns

Mode: setup analysis insertion

Usage

report_ijtag_instances [*name_spec*] [-below_instances *below_instances*]
[-filter *filter*] [-limit *limit*]
[-silent]

Description

Reports options of instances for which there is a matched ICL module.

You can restrict the report to instances below a given collection of instances and filter the collection based on an attribute filtering equation involving other attributes registered against or inherited by the instance object type.

The -silent option suppresses the warning that is normally generated if there are no Ijtag instances to report.

Arguments

- ***name_spec***

  An optional string or Tcl list of strings that specifies one or more patterns to be used to filter the returned list of instances. The string is a Tcl list of patterns separated by spaces and enclosed in braces {}. If no name_patterns are specified, the tool searches all instances.

  If you specify the -below_instances options, the name patterns are hierarchical name patterns relative to the specified instance objects; otherwise the name patterns are relative to the current design.

- -below_instances *instance_objects*

  An optional switch and value pair that constrains the command to search for instances below the specified instance_objects. The specified name_patterns are searched relative to each instance found in the instance_objects list.

- -filter *attribute_equation*

  An optional switch and string that specify to filter results based on the expression specified by the *attribute_equation* string. The attributes used within the equation must exist for the instance object type. See section "Attribute Filtering Equation Syntax" on page 105 for more details on filtering attribute equation format.

- -silent

  An optional switch that suppresses the warning message normally generated when the command does not find any instances to report. The warning message is intended to indicate a probable mistake when specifying patterns. You can use the -silent switch to suppress the warning. This -silent option has no effect on any other error reporting.

Example

The following example uses the report_ijtag_instances command to report the options of the first three instances of the Ijtag module. The command create_dft_specification provides access to those features if they are not already connected to the network.

```
> report_ijtag_instances -l 3
//
// ijtag instance: pb_di_do
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 4
// data_out_port_count      : 0
//
// ijtag instance: pb_di_do/dido_bus
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 16
// data_out_port_count      : 16
//
// ijtag instance: pb_si
// ----------------------------------------------------------------
// use_in_dft_specification : auto
// scan_interface_count     : 1
// data_in_port_count       : 0
// data_out_port_count      : 0
//
// Reached limit of 3, skipping remaining 9 instances.
```

Related Commands

get_ijtag_instances

get_ijtag_instance_option

set_ijtag_instance_options

report_ijtag_logical_connections

Context: patterns -ijtag

Mode: setup, analysis

Usage

report_ijtag_logical_connections [-from *source_pin_or_port_name*]
[-to *destination_pin_or_port_name*]

Description

Reports the logical paths that exist within the current design between the specified source and destination pins/ports, as well as all connections from or to the specified pins/ports. If no pin or port connections are specified, all logical connections are listed.

Arguments

- -from *source_pin_or_port_name*

  An optional switch and value pair that specifies the source pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

- -to *destination_pin_or_port_name*

  An optional switch and value pair that specifies the destination pin/port for the logical connections, which should be a top level module port or a hierarchical instance pin in the current top design module.

Example 1

The following example will list the logical connections between the pins /i1/in and /i32/u11/x10, as well as all logical connections from /i1/in and all connections to /i32/u11/x10.

report_ijtag_logical_connection -from /i1/in -to /i32/u11/x10

```
//  IJTAG Logical Connections:
//  From Source  To Destination
//  ===========  ==============
//  /i1/in       /i32/u11/x10
```

Example 2

The following example will list all the logical connections in the current design.

add_ijtag_logical_connection -from /i2/in -to /i2/out

add_ijtag_logical_connection -from /i3/in -to /i3/out

add_ijtag_logical_connection -from /i4/in -to /i4/out

report_ijtag_logical_connections

```
//  IJTAG Logical Connections:
//  From Source  To Destination
```

```
//  ===========  ==============
//  /i2/in       /i2/out
//  /i3/in       /i3/out
//  /i4/in       /i4/out
```

Related Commands

add_ijtag_logical_connection

delete_ijtag_logical_connection

report_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Usage

report_ijtag_retargeting_options

Description

Reports the type and value of each option of the previous set_ijtag_retargeting_options command or their default values if the set_ijtag_retargeting_options command was not issued.

To use this command, you must have previously loaded and elaborated the top-level ICL module. Otherwise, the tool issues an error message.

Example

The following example loads the ICL, turns off the reset of the ICL network after the last test_setup PDL command then reports the type and value of all three options:

SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl ../data/chip.icl
SETUP> set_current_design

```
//  Note: Top design is 'chip'.
```

SETUP> report_ijtag_retargeting_options

```
//  Option                          | Type | Value
//  --------------------------------+------+------
//  test_setup_network_end_state    | ENUM |  keep
//  compare_constant_capture_values | BOOL |    on
//  compare_constant_capture_values | BOOL |   off
```

Related Commands

get_ijtag_retargeting_options

set_ijtag_retargeting_options

report_input_constraints

Context: all contexts

Mode: all modes

Usage

Context: dft -scan and dft-test_points

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent] [***constraint_switch***] [-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]

Context: patterns -scan

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude | -SCAN_INputs | -EQuivalent] [***constraint_switch***] [-HOld] [-NO_Z] [-SLow_pad [-CELL *model_name*]] [-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]

Context: patterns -ijtag

report_input_constraints
[-ALl | *primary_input_pin_name*… | -BIDI_Only | -BIDI_Exclude| -EQuivalent] [***constraint_switch***] [-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]

Description

Displays the present constraint status of primary input pins.

The report_input_constraints command displays the constraint status of primary input pins to which you have applied constraints using the add_input_constraints command.

Arguments

- -ALl

  An optional switch that specifies to display the current constraint status of every primary input pin to which you previously applied an input constraint using the add_input_constraints command. Pins from which you removed the constraint with the delete_input_constraints command prior to reporting the status are included in the report.

- *primary_input_pin_name*

  An optional, repeatable string that specifies the name of a primary input pin for which to display the constraint status.

- -BIDI_Only

  An optional switch that specifies to display the constraint status of bidirectional pins only.

- -BIDI_Exclude

  An optional switch that specifies to display the constraint status only of primary input pins that are not bidirectional.

- -SCAN_INputs

  An optional switch that specifies to display the constraint status of scan input pins only.

- -EQuivalent

  An optional switch that specifies to display the constraint status of primary input pins that have a pin equivalence definition applied to them. These are pins for which you used the -Equivalent switch with the add_input_constraints command. The -C_all constraint switch is the only constraint switch you can include in the same command line with -Equivalent.

- *constraint_switch*

  An optional switch that specifies to report only on primary input pins that have a particular type of constraint applied. The choices, from which you can specify only one, are as follows:

  -C0 — Constant 0

  -C1 — Constant 1

  -CX — Constant X (unknown)

  -CZ — Constant Z (high-impedance)

  -CR0 — Constant that returns to 0

  -CR1 — Constant that returns to 1

  -CT0 — Constant TIE0

  -CT1 — Constant TIE1

  -CTZ — Constant TIEZ

Note

For additional information about the constraints associated with the preceding switches, refer to the same switches under the add_input_constraints command.

  -C_All — All C-type constraints

- -HOld

  An optional switch that specifies to report only input constraints that currently exhibit hold behavior. For information about what hold behavior is, refer to the -Hold argument under the add_input_constraints command.

- -NO_Z

  An optional switch that specifies to report only input constraints that currently exhibit no-Z behavior. For information about what no-Z behavior is, refer to the -No_z argument under the add_input_constraints command.

- -SLow_pad

  An optional switch that specifies to report only input constraints that currently exhibit slow-pad behavior. For information about what slow-pad behavior is, refer to the -Slow_pad argument under the add_input_constraints command.

- -CELL *model_name*

  An optional switch and string pair that, together with the -Slow_pad switch, specifies to prefix the instance name of each instance of the ATPG library model, *model_name*, to the *primary_input_pin_name* and look up each resulting name as the pin on which to report.

- -Display {[DEBug] [DESign] [DAta] | ALl} [APpend]

  An optional switch, repeatable literal, and literal that specify to display the primary input(s) in the specified window. The "Append" option adds the primary inputs to the existing contents of the window instead of replacing the contents. The window choices are as follows:

  DEBug
  Specifies the Debug window. This is the invocation default.

  DESign
  Specifies the Design window.

  DAta
  Specifies the Data window.

  ALl
  Specifies all supported windows (Debug, Design, and Data).

Example

The following example reports input constraints for a primary input.

report_input_constraints indata2

```
//  Primary Input  C-Type  Hold      No-Z      Slow-Pad
//  -------------  ------  ----  ------------  --------
//  indata2          C0     No   Undetermined     No
```

Related Commands

add_input_constraints
delete_input_constraints

report_layout_rules

Context: patterns -scan_diagnosis

Mode: analysis

Usage

report_layout_rules [-Fails_summary | -Summary | *layout_rule_id*... | *layout_rule_id-occurrence#*... | -All_fails | -MISmatch_report | -NO_LAYout_names] {[-INClude ***design_module_name***...] | [-EXClude ***design_module_name***...]} [{> | >>} *file_pathname*]

Description

Provides detailed reporting of layout rule violations.

Before using this command, you must run layout verification as part of the create_layout command or use open_layout with an existing Layout Database.

Note

Use this command in conjunction with the create_layout command to debug layout rule violations before you create the diagnosis Layout Database.

See "Layout-Aware Diagnosis and Reporting" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -Fails_summary

  A switch that specifies to display the following for each user-controllable rule that resulted in a violation (fail) during layout verification:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule
  - Brief description of the rule

  This is the default.

- -Summary

  A switch that specifies to display a summary report of layout rule violations. When you report on all rules (report_layout_rules -Summary), the following is reported for each user-controllable rule, whether or not it resulted in a violation (fail) during layout verification:

  - Rule identification (ID)
  - Number of failures of the rule
  - Current handling status of the rule

  - Brief description of the rule

- *layout_rule_id*

  A repeatable string that specifies the identification literal (ID) of a particular layout rule for which you want to display the violations.

- *layout_rule_id-occurrence#*

  A repeatable string that specifies the identification literal (ID) of a particular layout rule and the violation occurrence for which you want to display the occurrence message. This argument must include the specific design rule ID (*rule_id*), the specific occurrence number of the violation, and the hyphen between them. Numbers to occurrences of layout rule violations are assigned as they are encountered; you cannot change the number assigned to a specific occurrence.

- -All_Fails

  A switch that specifies to display all occurrence messages for all occurrences of layout rule violations. The displayed information can be lengthy, as it is the same information you would get if you consecutively entered a "report_layout_rules <*layout_rule_id*>" command for each rule that had a violation. Use this switch to output a report of all violation occurrences (most likely to a log file) for later analysis.

- -MISmatch_report

  An optional switch that re-generates the layout verification rule violation report. Normally, you use this switch in conjunction with the report_layout_rules command's -INClude and -EXClude switches to filter the layout rule violations—see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

- -NO_LAYout_names

  An optional switch that suppresses the reporting of instance/net/pin path names in layout format for selected layout rule violations. See "Instance, Net, and Pin Path Names in Layout Rule Violation Reports" in the *Tessent Diagnosis User's Manual* for complete information.

- -INClude ***design_module_name***

  A switch and repeatable string that defines an excluded area of the design's hierarchy when performing layout-to-design verification. The excluded area comprises all design elements that are *not* below instances of the included modules.

  The ***design_module_name*** is a space-delimited list of design modules. For example:

  report_layout_rules -include moduleA moduleB moduleD

  The -INClude and -EXClude switches are mutually exclusive.

  For detailed usage of this switch, see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

- -EXClude ***design_module_name***

  A switch and repeatable string that defines an excluded area of the design's hierarchy when performing layout-to-design verification. The excluded area comprises all design elements contained below instances of the excluded modules.

  The ***design_module_name*** is a space-delimited list of design modules. For example:

  report_layout_rules -include moduleA moduleB moduleD

  The -INClude and -EXClude switches are mutually exclusive.

  For detailed usage of this switch, see "Guidelines for Including or Excluding Design Modules From Mismatch Reporting" in the *Tessent Diagnosis User's Manual*.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

report_layout_rules

```
// Warning: Rule DesignCellPinMatch violated 1 times out of 226 checks.
// Warning: Rule DesignInstanceMatch violated 48 times out of 5344
// checks.
// Warning: Rule DesignNetPinMatch violated 139 times out of 18630 checks.
// Warning: Rule LayoutNetPinMatch violated 66 times out of 18557 checks.
// Warning: Rule PinExistenceMacro violated 3 times out of 44202 checks.
```

Without any switches a summary of all layout rule violations is printed. In addition to the number of violations, the total number of times a particular rule was checked is also printed. As an example, for the DesignInstanceMatch rule above, this rule was checked 5344 times (i.e. there must be 5344 library cell instances in this design) and it was violated 48 out of these 5344 checks.

Example 2

report_layout_rules PinExistenceMacro

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist
in macro).
// Warning: PinExistenceMacro-1: The specified NET cpu_i/uOPD/nx104 is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
// Warning: PinExistenceMacro-3: The specified NET cpu_i/INTR_PRI1_0_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
```

By specifying a rule name all the violations of that rule will be printed. First a summary for the user specified rule is printed and then each rule violation is printed in detail.

Example 3

report_layout_rules PinExistenceMacro-2

```
PinExistenceMacro: #fails=3 #checks=44202 handling=warning (pin not exist
in macro).
// Warning: PinExistenceMacro-2: The specified NET cpu_i/INTR_PRI1_7_ is
// connected to MACRO oai222 which uses a PIN C0 which is not defined in
// the LEF files.
```

This prints one particular violation of a given rule.

Example 4

In this case the instance name in DFT format differs from that in layout format since the last part of the name is flattened. Since the layout is being created from the LEF/DEF the instance path name in layout format is expected to be as follows:

```
/core0/cntl_1/mac\/adder\/AND0
```

Note that in LEF/DEF naming convention individual hierarchy delimiters are escaped.

Report the corresponding DesignInstanceMatch rule violation using the following command:

report_layout_rules DesignInstanceMatch-1

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0
(Expected Layout Name: /core0/cntl_1/mac\/adder\/AND0) not found in
layout.
```

You can suppress reporting of the layout name for this violation using the report_layout_rules command as follows:

report_layout_rules DesignInstanceMatch-1 –no_layout_names

```
// DesignInstanceMatch-1: Design instance /core0/cntl_1/mac/adder/AND0 not
found in layout.
```

Related Commands

create_layout

open_layout

report_lbist_configuration

Context: dft -logic_bist

Mode: setup, analysis

Usage

report_lbist_configuration [> | >> *file_pathname*]

Description

Reports the global logicBIST controller configuration parameters.

To report the parameters for individual EDT/LogicBIST blocks, use the report_edt_configurations command.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The example shows how to configure the LBIST controller parameters and then report on what has been specified.

```
set_lbist_controller_options -max_shift 400 -max_capture 3 -max_pattern 100000 \
        -capture {clk1P 30 clk2P 60 rstP 10}

report_lbist_configuration

//  IP version:                     1
//  Max Shift cycles:               400
//  Max Capture cycles:             3
//  Max pattern count:              100000
//  Capture procedures:             clk1P 30%
//                                  clk2P 60%
//                                  rstP 10%
```

Related Commands

report_edt_configurations

set_lbist_power_controller_options

report_lbist_pins

Context: dft -logic_bist

Mode: setup, analysis

Usage

report_lbist_pins [> | >> *file_pathname*]

Description

Reports the LogicBIST controller pins using the set_lbist_pins command.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

This example reports all the design pins to which the logicBIST controller's control signals, including the TAP interface signals are connected.

report_lbist_pins

```
//
//  PinType                 Connection
//  -------                 ----------
//  Clock                   refclk
//  Scan enable             scan_en
//  TCK                     tck
//  Xbounding enable        lbist_en
//  Setup Shift Scan in     tdi
//  Setup Shift Scan out    tdo jtag/scanCfgReg_so
//  Shift DR                - jtag/shift_dr
//  Capture DR              - jtag/capture_dr
//  Update DR               - jtag/update_dr
//  Test logic reset        - jtag/tlr
//  TAP instruction decode  - jtag/scanCfgReg_en
```

Related Commands

set_lbist_pins

report_edt_pins

report_lfsr_connections

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_lfsr_connections

Description

Displays a list of all the connections between Linear Feedback Shift Registers (LFSRs) and primary pins.

The report_lfsr_connections command displays all of the connections between the LFSRs and the primary pins. These connections were specified with the add_lfsr_connections commands.

Example

The following example displays the connections between the LFSRs and primary pins:

```
add_lfsrs lfsr1 prpg 5 15 -serial -in
add_lfsrs lfsr2 prpg 5 13 -serial -in
add_lfsrs misr1 misr 5 11 -parallel -in
add_lfsr_taps lfsr1 1 3
add_lfsr_taps lfsr2 2 3
add_lfsr_connections scan_in.1 lfsr1 3
add_lfsr_connections scan_out.0 misr1 2
report_lfsr_connections
```

Related Commands

add_lfsr_connections

delete_lfsr_connections

report_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_lfsrs

Description

Displays a list of definitions for all the current Linear Feedback Shift Registers (LFSRs).

The report_lfsrs command displays all of the LFSRs with their current values and tap positions that you specified using the add_lfsrs and add_lfsr_taps commands.

Example 1

The following example displays the definitions of all the current LFSRs:

add_lfsrs lfsr1 prpg 5 15 –serial -in
add_lfsrs lfsr2 prpg 5 13 –serial -in
add_lfsrs misr1 misr 5 11 –parallel –in
report_lfsrs

Example 2

The following example defines a 24-bit PRPG with self-feedback cells at positions 3, 7, 12, 15, and 23 and then lists the LFSRs:

add_lfsrs my_prpg prpg 24 FFFFFFFF -CA
add_lfsr_taps my_prpg 3 7 12 15 23
report_lfsrs

Output:

```
list of LFSRs
my_prpg  PRPG  length=24  shift=serial  ca_taps=3,12,15,23
my_prpg_pre_unload_value = 111111111111111111111111
my_prpg_post_unload_value = 111111111111111111111111
```

Related Commands

add_lfsrs
add_lfsr_taps
delete_lfsrs
set_lfsrs

report_licenses

Context: unspecified, all contexts

Mode: all modes

Usage

report_licenses

Description

Reports the license features currently checked out by the tool.

Arguments

- None

Example

The following example displays the licenses currently being used by the automatic diagnosis session.

report_licenses

The following report displays:

```
Licenses:
4 yieldascandiag
1 yieldaylearn
```

report_lists

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_lists [{> | >>} *file_pathname*]

Description

Displays the list of pins whose values the tool will report during simulation.

The report_lists command displays pins the tool is currently set up to report on during simulation. Use the add_lists command to add pins to the list and the delete_lists command to remove pins from the list.

When you switch to Setup mode, the tool discards all pins from the list.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example displays the pins whose values the tool will report during simulation:

add_lists /i_1006/o /i_1007/o
report_lists

```
2 pins are currently monitored.
i_1006/o
i_1007/o
```

Related Commands

add_lists
delete_lists
set_list_file

report_loops

Context: all contexts

Mode: all modes

Usage

report_loops [-All | *loop_id#…*] [-Display {DESign [-Hierarchical] | DAta | ALl} [APpend]] [{> | >>} *file_pathname*]

Description

Displays information about circuit loops.

The report_loops command displays information about currently identified loops in the circuit. For each loop, the report indicates whether the loop was broken by duplication. The report shows loops unbroken by duplication to be broken instead by a constant value, which means the loop is either a coupling loop or has a single multiple-fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

You can write the loops report information to a file by using the command's redirection operators or the write_loops command.

Arguments

- -ALl

  An optional switch that specifies to report all the loops in the circuit. This is the default.

- *loop_id#*

  An optional, repeatable, positive integer that specifies the identification number of a particular loop to report. The tool assigns loop identification numbers consecutively, starting with 1.

- -Display {DESign [-Hierarchical] | DAta | ALl} [APpend]

  An optional switch, repeatable literal, and literal that specify to display the primary input(s) in the specified window; if *Append* is specified, the primary inputs are added to the existing contents of the window instead of replacing the contents.

  The choices are as follows:

  DESign — A literal that specifies the Design window. If the *-Hierarchical* literal is specified, only the top-level hierarchical instances are displayed in the Design window; if the *-Hierarchical* literal is not specified, the child instances of the corresponding hierarchical instances are also displayed.

  DAta — A literal that specifies the Data window.

  ALl — A literal that displays the information in all the preceding windows.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example displays a list of all the loops in the circuit:

set_system_mode analysis
report_loops

```
Loop = 1: not_duplicated (coupling loop)
    my_design/my_minibus (SBUS)
    my_design/PAD (BUF)
    my_design/my_minibus (Z2X)
Loop = 2: not_duplicated (coupling loop)
...
Loop = 8: not_duplicated (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 2

The following example displays loop 8 only:

report_loops 8

```
Loop = 8: not_duplicated (single multiple fanout)
    my_design/al/pl/padx (BUF)
    my_design/al/pl/pad (BUF)
    my_design/pad (WIRE)
```

Example 3

The following example writes the display information for loop 8 to a new file named *my_loop_file*:

report_loops 8 > my_loop_file

```
... writing to file my_loop_file
```

Related Commands

report_feedback_paths

write_loops

report_lpct_configuration

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_lpct_configuration [> | >> ***filename***]

Description

Reports the current low pin count test controller (LPCT) parameters to the standard display output.

After saving the patterns, you can use the report_register_value command to report the different fields of the test configuration registers such as shift_length, capture_depth, scan_pattern_count, and chain_pattern_count.

For more details on the values reported by this command, see the set_lpct_controller command.

Arguments

- > ***file_pathname***

  An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of ***file_pathname*** with command output.

- >> ***file_pathname***

  An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of ***file_pathname***.

Example 1

The following example shows a report generated from Setup mode during EDT logic creation:

report_lpct_configuration

```
// Max Shift cycles:              1000
// Max Capture cycles:            8
// Max Scan patterns:             1048575
// Max Chain patterns:            1023
// Test mode detect sequence:     1010000101001
// Test mode detect period:       100 cycles
// Scan shift control:            internally generated clock
```

Related Commands

report_lpct_pins

set_lpct_controller

report_lpct_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

report_lpct_pins [{> | >>} ***filename***]

Description

Reports the low pin count test (LPCT) controller pins and connection information to the standard display output.

Arguments

- > ***file_pathname***

  An optional redirection operator and pathname pair used at the end of an argument list to create or replace the contents of ***file_pathname*** with command output.

- >> ***file_pathname***

  An optional redirection operator and pathname pair used at the end of the argument list to append the command output to the contents of ***file_pathname***.

Example

The following example shows a report generated from Setup mode during EDT logic creation:

report_lpct_pins

```
//
//  Input pin            Pin name        Inversion  Internal connection
//  ---------            --------        ---------  -------------------
//  Clock                port_clock          -               -
//  Reset (Active High)        -             -      pwr_up/reset
//  Data in              lpct_data_in        -               -
//  Test mode            lpct_test_mode      -               -
//
//  Output pin        Connection
//  ----------        ----------
//  Capture enable
//  Clock mux select
//  Shift clock
//  Scan enable       scan_en
//  Reset out
//  Test end
//  Data out
//  Test active
//
```

Related Commands

report_lpct_configuration

set_lpct_controller

report_measure_cycles

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

Mode: analysis

Usage

report_measure_cycles [-All_cycles] [-Suite *test_suite_ID*] [{-Pattern_range | -Cycle_range} *begin end*] | [-List *cycles* …] [*observe_point*…] [> | >> *filename*]

Description

Displays data for the specified components/cycles/patterns from the test patterns loaded for the current session. If nothing is specified, data for all the test patterns is reported.

Depending on the arguments used with this command, you can report expect bits, masked bits, cycle padding, cycles, patterns, and chain cells and lengths.

You can use this command to troubleshoot data verification errors that occur before diagnosis. Only WGL and STIL test patterns are supported by this command.

Use this command to debug failure file mismatches reported by the tool. You can use this command to report the failing cycle, failing pattern, expect values, and boundary cycles related to a specific mismatch.

Arguments

- -All_cycles

  An optional switch that generates a full report for all cycles.

  The tool assigns each cycle to a specific category, and groups the neighboring cycles belonging to the same category into a cycle block, which include one or more cycles.

  For each cycle block, the tool reports the following in columnar format:

  - **First** — First cycle.
  - **Last** — Last cycle (unless the cycle is the same as the first cycle).
  - **#Cycle** — Count of cycles.
  - **Pid** — Pattern ID.
  - **Type** — Pattern type.
  - **Sid** — Suite ID. Reported if there is more than one test suite.
  - **RelPid** — Relative Pattern ID. Reported if there is more than one test suite.
  - **Operation** — Function of the cycle block, for example “Test setup”, “Setup shift”, “Force PI”, “Measure PO” and “Shift and measure SCO”.

- -Suite *test_suite_ID*

  An optional switch and string pair that specifies the targeted test suite when you use more than one test suite. When you specify a valid test suite ID, the pattern IDs (begin end) specified are treated as relative IDs for this targeted test suite. Otherwise, the pattern IDs (begin end) are considered as absolute numbers.

  By default, if you omit this switch, then the pattern IDs (begin end) are considered as absolute numbers. If only the -suite switch is used and the -pattern is omitted, then the tool reports all patterns of the specified test suite.

- -Pattern_range *begin end*

  Optional switch and value pair that specifies a range of patterns.

- -Cycle_range *begin end*

  Optional switch and value pair that specifies a range of cycles. This option cannot be used with -All_cycles because a full cycle-oriented report is generated; the tool ignores the -all_cycles option, if supplied.

- -List *cycles*

  Optional, repeatable switch and value pair used to specify one or more specific cycles.

- *observe_point*

  Optional value that specifies the name of one or more components to focus the report on. Observe points are usually pin names, but may also represent any of the following:

  - chain
  - scanout_pin
  - edt_channel
  - edt_scanout_pin
  - POpin

  For a compressed pattern design, if an internal scan chain is used as an observe_point, no value is reported. The observe point for a compressed pattern should be either edt channel, edt channel output pin, or primary output.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*. Use this argument to write the report to a file.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*. Use this argument to append the report to the specified file.

Example 1

The following example displays a specified range of patterns:

report_measure_cycles -pattern 10 20

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern   Measure_PO  Measure_SCO (first,last,step)
 ---------  ----------  ----------------------------
     10 (S)   2277    { (2279,2380,+1) }
     11 (S)   2381    { (2383,2484,+1) }
     12 (S)   2485    { (2487,2588,+1) }
     13 (S)   2589    { (2591,2692,+1) }
     14 (S)   2693    { (2695,2796,+1) }
     15 (S)   2797    { (2799,2900,+1) }
     16 (S)   2901    { (2903,3004,+1) }
     17 (S)   3005    { (3007,3108,+1) }
     18 (S)   3109    { (3111,3212,+1) }
     19 (S)   3213    { (3215,3316,+1) }
     20 (S)   3317    { (3319,3420,+1) }
```

Example 2

The following example displays the cycles, patterns, and expect bits information for the chain test for chain3:

report_measure_cycles chain3

```
total cycles = 41103
cycle  pattern  chain{cell}:scanout  expect
-----  -------  -------------------  ------

105     0       chain3{0}:scan_out3 L  chaintest
106     0       chain3{1}:scan_out3 L  chaintest
107     0       chain3{2}:scan_out3 H  chaintest
108     0       chain3{3}:scan_out3 H  chaintest
109     0       chain3{4}:scan_out3 L  chaintest
110     0       chain3{5}:scan_out3 L  chaintest
111     0       chain3{6}:scan_out3 H  chaintest
112     0       chain3{7}:scan_out3 H  chaintest
113     0       chain3{8}:scan_out3 L  chaintest
114     0       chain3{9}:scan_out3 L  chaintest
115     0       chain3{10}:scan_out3 H  chaintest
116     0       chain3{11}:scan_out3 H  chaintest
117     0       chain3{12}:scan_out3 L  chaintest
118     0       chain3{13}:scan_out3 L  chaintest
119     0       chain3{14}:scan_out3 H  chaintest
120     0       chain3{15}:scan_out3 H  chaintest
```

Example 3

The following example reports all cycles based on cycle blocks.

report_measure_cycles -all_cycles

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )

First  Last   #Cycle  Pid  Type  Operation

0             1       0    C     Test setup
1             1       0    C     Setup shift
2      103    102     0    C     Shift (load only)
104           1       0    C     Setup shift
105    206    102     0    C     Shift and measure SCO
207           1       1    C     Setup shift
208    309    102     1    C     Shift and measure SCO
310           1       2    C     Setup shift
311    412    102     2    C     Shift and measure SCO
413           1       3    C     Setup shift
414    515    102     3    C     Shift and measure SCO
516           1       4    C     Setup shift
517    618    102     4    C     Shift and measure SCO
619           1       5    C     Setup shift
620    721    102     5    C     Shift and measure SCO
722           1       6    C     Setup shift
723    824    102     6    C     Shift and measure SCO
825           1       7    C     Setup shift
826    927    102     7    C     Shift and measure SCO
928           1       8    C     Setup shift
929    1030   102     8    C     Shift and measure SCO
1031          1       9    C     Setup shift
1032   1133   102     9    C     Shift and measure SCO
1134          1       0    S     Setup shift
1135   1236   102     0    S     Shift (load only)
1237          1       0    S     Measure PO and pulse clock
1238          1       0    S     Setup shift
1239   1340   102     0    S     Shift and measure SCO
1341          1       1    S     Measure PO and pulse clock
```

Example 4

The following example reports the measurement cycles for test suite 1.

report_measure_cycles -suite 1

```
total cycles = 82206
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
pattern set # 2 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )
  Pattern (Suite/Pid)  Measure_PO  Measure_SCO (first,last,step)
 --------------------  ----------  -----------------------------
      0 ( 1/   0) (C)   ----    { (105,206,+1) }
      1 ( 1/   1) (C)   ----    { (208,309,+1) }
      2 ( 1/   2) (C)   ----    { (311,412,+1) }
      3 ( 1/   3) (C)   ----    { (414,515,+1) }
      4 ( 1/   4) (C)   ----    { (517,618,+1) }
      5 ( 1/   5) (C)   ----    { (620,721,+1) }
      6 ( 1/   6) (C)   ----    { (723,824,+1) }
      7 ( 1/   7) (C)   ----    { (826,927,+1) }
```

```
8 ( 1/   8) (C)   ----    { (929,1030,+1) }
9 ( 1/   9) (C)   ----    { (1032,1133,+1) }
0 ( 1/   0) (S)   1237    { (1239,1340,+1) }
1 ( 1/   1) (S)   1341    { (1343,1444,+1) }
2 ( 1/   2) (S)   1445    { (1447,1548,+1) }
3 ( 1/   3) (S)   1549    { (1551,1652,+1) }
4 ( 1/   4) (S)   1653    { (1655,1756,+1) }
5 ( 1/   5) (S)   1757    { (1759,1860,+1) }
6 ( 1/   6) (S)   1861    { (1863,1964,+1) }
```

Example 5

The following example reports in blocks, all cycles between 0 and 200, and shows that cycles 105-206 are measure cycles for the first chain pattern.

report_measure_cycles -cycle_range 0 200

```
total cycles = 41103
pattern set # 1 total cycles: 41103 last chain: 9 last scan: 382
( pat/pat.wgl )

First  Last   #Cycle  Pid  Type  Operation

0             1       0    C     Test setup
1             1       0    C     Setup shift
2      103    102     0    C     Shift (load only)
104           1       0    C     Setup shift
105    206    102     0    C     Shift and measure SCO
```

Related Commands

diagnose_failures
read_patterns
set_diagnosis_options

report_mismatch_sources

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_mismatch_sources [*src-id* | -ALl]
[-Display [DEBug] [DESign] [DAta] [Wave] | ALl]
[(> | >>) *file_pathname*]

Description

Displays the sources of simulation mismatches.

The report_mismatch_sources command displays detailed information about the sources of simulation mismatches found by a previous analyze_simulation_mismatches command or a write_patterns -Debug command. For each mismatch source you specify, the command reports the following information in columnar format:

- ID# — Positive integer that identifies a specific mismatch source. The tool numbers mismatch sources consecutively, starting at 1.
- Instance name — Hierarchical instance name of the mismatch source.
- Pattern number — The number of the first failing pattern caused by a mismatch source. In the ModelSim Wave window, the pattern number is the value of the "_pattern_count" variable at the time the mismatch occurred. This variable is displayed in the top line of the Wave window and increments as each pattern is simulated. This number corresponds to the pattern number in ASCII patterns.
- Time (ns) — First simulation time where the source causes a value in the external simulator that is inconsistent with what the tool expected.
- Capture cycle — The capture cycle where the first mismatch occurs (starting from 0).
- Expected value (ATPG_val) — Expected output value from the tool.
- Simulated value (VCD_val) — Simulated output value from the external simulator. Possible values are:

  0 — Logic 0

  1 — Logic 1

  Z — Tristate

  X — Unknown

  U — Unused

  . — No VCD data was saved.
- Reason — The following is a list of possible mismatch reasons:

- “incorrect input”: Mismatch is due to wrong force values at PI.
- “incorrect PO measure”: Mismatch is due to wrong measure PO time.
- “incorrect loading”: Mismatch is due to incorrect scan load.
- “incorrect capture”: Mismatch is due to incorrect capture clock even at the state element.
- “incorrect unloading”: Mismatch is due to incorrect scan unload.
- “wrong clock pulse”: Mismatch is due to incorrect clock pulse.
- “setup time violation”: Mismatch is due to setup time violation, i.e. combinational path is not stable in one clock cycle.
- “incorrect logic”: Mismatch is due to difference logic between ATPG model and the Verilog model. This includes different netlists used between ATPG tool and Verilog simulator, or ATPG library model difference to the corresponding Verilog model.
- “mismatched RAM model”: Mismatch is due to difference on RAM model between ATPG library and Verilog library.
- “incorrect library cell”: Mismatch is due to difference between ATPG library model and Verilog library cell.
- “incorrect ATPG setting”: Mismatch is due to improper setting on ATPG tool. Most common mistakes are incorrect setting of set_split_capture and set_clock_off_simulation.
- “hold time violation”: Mismatch is due to hold time violation. The sink flop of the hold time violation is reported as the mismatch source gate.
- “clock/data port race”: mismatch is due to clock and data racing. Common clock racing situations include C6 and C4 violation.

Arguments

- *src-id*

  An optional positive integer that specifies the identification number of the mismatch source to report. The tool assigns source IDs to mismatch sources consecutively, starting with 1.

- -Display [DEBug] [DESign] [DAta] [Wave]

  An optional switch that specifies to display the specified source(s) in the Debug, Design, Data, or Wave window of DFTVisualizer. If the *Display* switch is specified without an argument, the Debug, Data, and Wave windows are displayed.

- -ALl

  An optional switch that specifies to display all the mismatch sources. This is the default.

Note

In the Debug window, pin names assigned to the value "." indicate that the VCD debug test bench did not capture the VCD value for that location. It is not possible to capture VCD values for every node in the design due to very large file sizes and run time; the test bench only captures values in the combination cone behind the failure location, which in most cases provides sufficient information to explain the mismatch.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example assumes the pathname of the ModelSim bin directory is defined by the PATH variable and patterns are generated but not yet saved. The example first specifies the invocation command for the external simulator, then saves the generated pattern set. Next, it creates the Verilog library *my_mismatch_work*, compiles the source for the top-level design and supporting simulation parts library into the library, and analyzes the simulation mismatches. Finally, DFTVisualizer is invoked and displays the simulation, test pattern, and design data with the mismatch sources highlighted in red.

```
set_external_ simulator vsim -c
save_patterns results pat_p.v -verilog -parallel -replace
sys vlib my_mismatch_work
sys vlog -work my_mismatch_work my_design_source/my_design.v -v \
      my_verilog_source/verilog_lib.v
```

```
Model Technology ModelSim SE vlog 5.5a Compiler 2001.04
-- Compiling module DFFS_scan
-- Compiling module my t op level module
-- Scanning library file 'my verilog source/verilog lib.v'
-- Compiling module xor3
-- Compiling module mux21
...
-- Compiling module buff
-- Compiling module mgc latch
-- Scanning library file 'my_verilog_source/verilog_lib.v'
-- Compiling UDP ud_dff
-- Compiling UDP ud_dlat

Top level modules:
      my_top_level_module
```

```
analyze_simulation_mismatches -auto -internal -lib my_mismatch_work
```

```
// New directory work_dft_debug has been created to store files for mismatch debug.
// Previous simulation mismatch sources are deleted.
//-----------------------------------------------------------------------------------------
//Step-1 Creating Verilog testbench work_dft_debug/mentor_default.v with failure file
//dump.
//-----------------------------------------------------------------------------------------
//Step-2 Running Verilog testbench work_dft_debug/mentor_default.v to create failure file
//-----------------------------------------------------------------------------------------
//Step-3 Loading failure file work_dft_debug/mentor_default.v.fail.
//-----------------------------------------------------------------------------------------
//Step-4 Creating Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v with VCD dump
//-----------------------------------------------------------------------------------------
//Step-5 Running Verilog testbench work_dft_debug/mentor_default.v_vcdtb.v to create VCD
//file.
//Running ModelSim to create VCD file work_dft_debug/mentor_default.v_debug.vcd
//-----------------------------------------------------------------------------------------
//Step-6 Loading VCD file work_dft_debug/mentor_default.v debug.vcd.
//-----------------------------------------------------------------------------------------
//Step-7 Analyzing 25 internal scan patterns in parallel format.
//-----------------------------------------------------------------------------------------
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason
//-- ------------------ ---------- -------- ------------- -------- ------- ----------------
// 1 /ix1286/Y (364)     scan-1       1900             0        1       0  incorrect lib cell "and4"
// 2 /ix1278/Y (265)     scan-15     17300             0        1       0  incorrect lib cell "and4"
```

report_mismatch_sources 2 -display debug data

DFTVisualizer opens and displays the following information about mismatches in the Transcript window; the sources are also displayed in the Debug and Data windows.

```
//ID instance (gate id) pattern-ix time(ns) Capture_cycle ATPG_val VCD_val mismatch_reason
//-- ------------------ ---------- -------- ------------- -------- ------- ----------------
// 2 /ix1278/Y (265)     scan-15     17300             0        1       0  incorrect lib cell "and4"
```

For more information about the Debug and Data windows, refer to "DFTVisualizer Window Overview" in the *Tessent Shell User's Manual*.

Related Commands

analyze_simulation_mismatches
set_external_simulator
write_patterns

report_misrs

Context: pattern -scan

Mode: setup, analysis

Usage

report_misrs {**-all** | ***misr_name*** …} [> | >> *file_pathname*]

Description

Reports the MISRs you have added to your design.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- **-all**

  A required switch that reports all MISRs.

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

This example reports all the MISRs defined across all the EDT/LogicBIST blocks.

report_misrs -all

```
list of MISRs
A_misr_0  MISR  length=24  in_stream_taps=20,21,23
  A_misr_0_pre_unload_value = 000000000000000000000000
A_misr_1  MISR  length=32  in_stream_taps=4,5,31
  A_misr_1_pre_unload_value = 00000000000000000000000000000000
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 00000000000100010001000100000000
A_misr_3  MISR  length=32  in_stream_taps=4,5,31
  A_misr_3_pre_unload_value = 00000000000000000000000000000000
B1_misr  MISR  length=24  in_stream_taps=20,21,23
   B1_misr_pre_unload_value = 000000000000000000000000
B2_misr_0  MISR  length=24  in_stream_taps=20,21,23
B2_misr_0_pre_unload_value = 000000000000000000000000
B2_misr_1  MISR  length=24  in_stream_taps=20,21,23
B2_misr_1_pre_unload_value = 000000000000000000000000
```

Example 2

This example reports a specific MISR: A_misr_2:

report_misrs A_misr_2

```
list of MISRs
A_misr_2  MISR  length=32  in_stream_taps=4,5,31
  A_misr_2_pre_unload_value = 00000000000100010001000100000000
```

Related Commands

add_misrs

delete_misrs

report_misr_connections

Context: patterns -scan

Mode: setup, analysis

Usage

report_misr_connections ***misr_name*** [> | >> *file_pathname*]

Description

Reports MISR connections.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***misr_name***

  A required repeatable string that specifies the name of the MISR.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

This example shows the connection between the LBIST scan chain output and the MISR input for a 24-bit MISR named 'misr'. The 16 scan chains are compacted with a 4-to-1 XOR and connected to the first 4 inputs of the MISR.

report_misr_connections misr

```
list of MISR connections
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3]  MISR=misr
connection_points=1
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6]  MISR=misr
connection_points=2
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7]  MISR=misr
connection_points=2
```

```
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11]  MISR=misr
connection_points=3
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14]  MISR=misr
connection_points=4
my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15]  MISR=misr
connection_points=4
```

Related Commands

add_misrs

set_misr_connections

report_module_matching

Context: dft, patterns -ijtag

Mode: all modes

Usage

report_module_matching **-icl**

Description

Reports design modules and their matching ICL modules if the set_current_design command has been executed in ICL extraction mode prior.

Arguments

- **-icl**

  A required switch that specifies to report matching ICL modules.

Return Value

None

Example

The following example shows an example report generated by this command.

```
read_icl ../data/icl/block3.icl
set_design_sources -format icl -search_design_load_path On
set_module_matching_options -suffix_pattern_list "_\[0-9\]" -regexp
set_current_design chip
report_module_matching -icl
```

```
Design     Design                   ICL         ICL
Module     Instance                 Module      File
-----------------------------------------------------------------
PLL_CORE   chip/pll1_I1/iPLL        PLL_CORE ../data/icl_prim/PLL_CORE.icl
block1_0   chip/block1_I1           block1   ../data/icl/block1.icl
block1_1   chip/block1_I2           block1   ../data/icl/block1.icl
block2     chip/block2_I1           block2   ../data/icl/block2.icl
block3     chip/block3_I1           block3   ../data/icl/block3.icl
pll1       chip/pll1_I1             pll1     ../data/icl_prim/pll1.icl
raw1_0     chip/block1_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_1     chip/block3_I1/raw1_I4   raw1     ../data/icl_prim/raw1.icl
raw1_2     chip/block3_I1/raw1_I3   raw1     ../data/icl_prim/raw1.icl
raw1_3     chip/block3_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_4     chip/block3_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_5     chip/block2_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_6     chip/block2_I1/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_7     chip/block1_I2/raw1_I2   raw1     ../data/icl_prim/raw1.icl
raw1_8     chip/block1_I2/raw1_I1   raw1     ../data/icl_prim/raw1.icl
raw1_9     chip/block1_I1/raw1_I2   raw1     ../data/icl_prim/raw1.icl
sib1_0     chip/sib1_IPLL           sib1     ../data/icl_prim/sib1.icl
sib1_1     chip/block1_I2/sib1_I2   sib1     ../data/icl_prim/sib1.icl
sib1_2     chip/block1_I2/sib1_I1   sib1     ../data/icl_prim/sib1.icl
sib1_3     chip/block1_I1/sib1_I2   sib1     ../data/icl_prim/sib1.icl
```

```
sib1_4     chip/block1_I1/sib1_I1  sib1     ../data/icl_prim/sib1.icl
sib1_5     chip/sib1_I4            sib1     ../data/icl_prim/sib1.icl
sib1_6     chip/sib1_I3            sib1     ../data/icl_prim/sib1.icl
sib1_7     chip/sib1_I2            sib1     ../data/icl_prim/sib1.icl
sib1_8     chip/sib1_I1            sib1     ../data/icl_prim/sib1.icl
tap1       chip/tap1_I1            tap1     ../data/icl_prim/tap1.icl
tdr1       chip/pll1_I1/tdr1_IPLL1 tdr1     ../data/icl_prim/tdr1.icl
tdr1_1     chip/block3_I1/tdr1_I4  tdr1     ../data/icl_prim/tdr1.icl
tdr1_2     chip/block3_I1/tdr1_I3  tdr1     ../data/icl_prim/tdr1.icl
tdr1_3     chip/block3_I1/tdr1_I2  tdr1     ../data/icl_prim/tdr1.icl
tdr1_4     chip/block3_I1/tdr1_I1  tdr1     ../data/icl_prim/tdr1.icl
tdr1_5     chip/block1_I2/tdr1_I2  tdr1     ../data/icl_prim/tdr1.icl
tdr1_6     chip/block1_I2/tdr1_I1  tdr1     ../data/icl_prim/tdr1.icl
tdr1_7     chip/block1_I1/tdr1_I2  tdr1     ../data/icl_prim/tdr1.icl
tdr1_8     chip/block1_I1/tdr1_I1  tdr1     ../data/icl_prim/tdr1.icl
tdr2       chip/block2_I1/tdr2_I1  tdr2     ../data/icl_prim/tdr2.icl
```

Related Commands

set_module_matching_options

report_module_matching_options

Context: all contexts

Mode: all modes

Usage

report_module_matching_options

Description

Reports the current settings defined by the set_module_matching_options command.

Arguments

None

Return Value

None

Example

The following example shows an example report generated by this command.

report_module_matching_options

```
//  command: read_icl ../data/icl/block3.icl
//  command: set_design_source -format icl -search_design_load_path On
//  command: set_module_matching_option -suffix_pattern_list "_mod_\[0-
9\]" -regexp
//  command: set_module_matching_option -prefix_pattern_list mgc -append
//  command: set_module_matching_option -suffix_pattern_list "_\[0-9\]" -
regexp -append
//  command: report_module_matching_options
//  affix_type  regexp  case  pattern
//  ----------  ------  ----  -------
//  prefix      no      yes   "mgc"
//  suffix      yes     no    "_mod_\[0-9\]"
//  suffix      yes     no    "_\[0-9\]"
```

Related Commands

set_module_matching_options

report_multicycle_paths

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

Mode: setup, analysis

Usage

report_multicycle_paths [-All] [-Error] [-Verbose] [-Debug_error]
[-From *source_node…* | -FROM_Clock *clock_signal_name…*]
[-TO *sink_node…* | -TO_Clock *clock_signal_name…*]
[{-THrough *through_node…*}…]
[{> | >>} *file_pathname*]

Description

Displays multicycle path definitions you previously read into the tool using the read_sdc command.

Note that you can display SDC filenames and line numbers by first issuing the "set_timing_exceptions_handling -line_no" command.

If a specified starting point, end point, source node, clock signal name, sink node, or through node does not translate to an internal gate, zero (0) multicycle paths are reported.

Arguments

Tip: When using the following arguments, you can include one or more asterisks (*) in pin or instance pathnames. The command treats an asterisk as a wildcard character, enabling you to use it to match many pathnames in the design.

- -All

  An optional switch that displays all previously-defined multicycle paths. This is the default.

- -Error

  An optional switch that displays only the multicycle paths that have errors and warnings.

- -Verbose

  An optional switch that displays all the gate information in the from, to, and through gate lists.

- -Debug_error

  An optional switch that displays all the previously-defined multicycle paths that have errors and warnings, along with their respective error and warning messages.

- -From *source_node…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path. A valid starting point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the output pins of the

instance are considered to be starting points. For a clock PI, the outputs of all the non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be starting points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

- -FROM_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the starting point of a multicycle path by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of a clock signal, and the outputs of all non-transparent latches, flip-flops, and RAMs associated with this clock signal are considered to be starting points.

- -TO *sink_node…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle path. A valid end point is a pin pathname, an instance pathname, a clock primary input (PI), or an internal clock pin name. For an instance pathname, all the input pins of the instance are considered to be end points. For a clock PI, the data inputs of all non-transparent latches, flip-flops, and RAMs associated with the clock PI are considered to be end points. A pin pathname is considered to be that of an internal clock pin if its fanout reaches only clock ports of sequential elements after traversing through any intervening combinational gates, and the internal clock pin is handled the same as a clock PI.

Note

For a 2-port latch with ports: Port1 = (D1, CLK1), Port2 = (D2, CLK2), and primary input clocks CLK1 and CLK2, the command “report_multicycle_paths... -to CLK2” would report multicycle paths with an end point at D2.

- -TO_Clock *clock_signal_name…*

  An optional switch and repeatable string pair that specifies the end point of a multicycle paths by its associated clock pin pathname. Any pin pathname you specify with this argument is interpreted to be the source of an internal clock signal and the data inputs of all non-transparent latches, flip-flops, and RAMs associated with this clock are considered to be end points.

- {-THrough *through_node…*}…

  An optional, repeatable switch and repeatable string pair that specifies circuit node(s) through which the multicycle path must pass. A valid through node is a pin or instance pathname. Clock PIs are not supported as through nodes and no attempt is made to determine if a pin pathname is that of an internal clock pin.

  When you use multiple -Through arguments, their order left-to-right in the command string is considered to be the order in which a signal would pass through the nodes on the specified path. For example, the following multicycle path specification:

  report multicycle paths -from A -through B C -through D E -to F

specifies all paths that start from A, pass through either B or C, then pass through D or E, and end at F. If you do not use the -Through argument when specifying a path, all paths from the specified starting points to the specified end points are considered to be multicycle paths.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example displays all previously-defined multicycle paths.

report_multicycle_paths -all

```
// Multicycle Path -from -from /cpu/d_in[7] -through /cpu/u1 -cycles 2
// Multicycle Path -to /cpu/c -cycles 2
//
// ---------------------------------------------------------------
// The multicycle path below has an error.
// ---------------------------------------------------------------
// Multicycle Path -from dummy1 -to dummy2 -cycles 2
//
// ---------------------------------------------------------------
// The multicycle paths below have warnings.
// ---------------------------------------------------------------
// Multicycle Path -from /cpu/d_in[7] -through /cpu/d_in_[6]
//     /cpu/u25 -cycles 2
// Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
// Multicycle Path -from CLK1 -through /cpu/xray[6] CLK1 -cycles 2
//
// Total reported multicycle paths = 6
// Warning: The last 4 paths reported have 1 error and 3 warnings.
//          Use the command "report_multicycle_paths -debug_error" to
//          report the causes of the errors/warnings.
// Warning: Multicycle path defined through all primary inputs, including
//          all clocks, which prohibits at-speed testing. Deleted clocks
//          from this list.
// Warning: There exists a multicycle path that prohibits at-speed testing
//          of clock domain /CLK1.
//          This may result in lower test coverage.
```

Example 2

The following example displays the multicycle paths with error and warning messages.

report_multicycle_paths -error

```
// Error: Multicycle Path -from dummy1 -to dummy2 -cycles 2
// Warning: Multicycle Path -through /cpu/d_in_[6]  /cpu/u25 -cycles 2
// Warning: Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
// Warning: Multicycle Path -from CLK1 -through /cpu/xray[6]
```

```
//          CLK1 -cycles 2
// Total reported multicycle paths = 4
```

Example 3

The following example displays debug information for the multicycle paths with error and warning messages.

report_multicycle_paths -debug_error

```
// Error: Multicycle Path -from dummy1 -to dummy2 -cycles 2 did not find
//        gates dummy1 dummy2
// Warning: Multicycle Path -from /cpu/d_in[7]-through /cpu/d_in[6]
//          /cpu/u25 -cycles 2 has no combinational path between start and
//          through points.
// Warning: Multicycle Path -through I1 I2 I3 I4 I5 SIN CLK1 SEN -cycles 3
//          defined through all primary inputs, including all clocks,
//          which prohibits at-speed testing.
//          Deleted clocks from this list.
// Warning: Multicycle Path -from CLK1 -through /cpu/xray[6] CLK1
//          -cycles 2 prohibits at-speed testing of clock domain /CLK1.
// Multicycle path debug summary: 1 error, 3 warnings.
```

Example 4

The following example displays a verbose report of multicycle paths that start from /cpu/d_in[7]. Notice the empty “( )” for the incorrectly defined path.

report_multicycle_paths -verbose -from /cpu/d_in[7]

```
// Multicycle Path -from /cpu/d_in[7] (/reg_d_1_7_/Q (81)) -through
//      /cpu/u1 (/reg_d_1_6_/Q (73)) -cycles 2
//
// ----------------------------------------
// The multicycle path below has an error.
// ----------------------------------------
// Error: Multicycle Path -from /cpu/d_in[7] ( ) -through /cpu/xray[6] ( )
//        -cycles 2
// Total reported false paths = 2
```

For examples of how to use the asterisk (*) character to specify multiple pathnames with one pathname string, see the add_false_paths command.

Example 5

The following shows multicycle path reporting after turning on reporting of SDC filenames and line numbers:

read_sdc sdc2
set_timing_exceptions_handling -line_no on
report_multicycle_paths

```
//  Multicycle Path -from sff108/Q -to sff209/D -cycles 3 -setup  (filename = sdc2, line 1)
```

Related Commands

add_false_paths
delete_false_paths
delete_multicycle_paths
read_sdc
report_false_paths
set_timing_exceptions_handling

report_multiprocessing_options

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Usage

report_multiprocessing_options

Description

Displays the values of all variables used when executing multiprocessing commands for ATPG or fault simulation.

The report_multiprocessing_options command displays in human-readable format the values of multiprocessing variables that affect the behavior of the tool's built-in multiprocessing manager. These are values that you can set with the set_multiprocessing_options command. To obtain the values of multiprocessing processing variables for Tcl scripting, use the get_multiprocessing_option command.

For more information, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "Using the Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

None.

Example

The following example displays the current settings of the multiprocessing variables;

report_multiprocessing_options

```
// Multiprocessing options:
// Option                        Type   Value   Description
// ---------------------------- ------ ------  -------------------------------------
// generic_delete               string         generic job scheduler delete
// generic_scheduler            string         generic job scheduler
// license_timeout              number  5      # mins to acquire license ( 0: infinite )
// lsf_heuristics               on/off on      true for LSF job scheduler heuristic learning
// lsf_learning                 on/off on      true for LSF job scheduler learning
// lsf_options                  string         options for LSF job scheduler
// multithreading               on/off on      turn on/off multithreading flow
// processors_per_grid_request number -1       # processors grouped for one grid request
//                                             (default: 1 for SGE and GENERIC, 4 for LSF)
// remote_shell                 string rsh     rsh or ssh remote_shell setting
// result_time_limit            float  45      time limit (min) used to detect non-responsive
//                                             slaves.
// scheduler_timeout            number 10      # mins for job scheduler
// sge_options                  string         options for SGE job scheduler
```

Related Commands

add_processors
delete_processors
get_multiprocessing_option
report_processors
set_multiprocessing_options

report_nofaults

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

Mode: setup, analysis

Usage

report_nofaults ***pathname***... | **-All** [-Instance] [-Stuck_at {01 | 0 | 1}] [-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

Displays the nofault settings for the specified pin pathnames or pin names of instances.

The report_nofaults command displays, for pin pathnames or pin names of instances, the nofault settings that you previously specified with the add_nofaults command.

Arguments

- ***pathname***

  A repeatable string that specifies the pin pathnames or the instance pathnames for which you want to display the nofault settings. If you specify an instance pathname, you must also specify the -Instance switch.

- **-All**

  A switch that displays the nofault settings on either all pin pathnames or, if you also specify the -Instance switch, all pin names of instances.

- -Instance

  An optional switch that specifies that the *pathname* or -All argument indicates instance pathnames.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition nofault settings you want to display. The choices are as follows:

  01 — A literal that specifies to display both the “stuck-at-0” and “stuck-at-1” nofault settings for stuck-at faults; or both the “slow-to-rise” and “slow-to-fall” nofault settings for transition faults. This is the default.

  0 — A literal that specifies to display only the “stuck-at-0” nofault settings (“slow-to-rise” nofault settings for transition faults).

  1 — A literal that specifies to display only the “stuck-at-1” nofault settings (“slow-to-fall” nofault settings for transition faults).

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the nofault settings which you want to display. The literal choices are as follows:

  Full — A literal that displays all the nofault settings in the user and system class. This is the default.

User — A literal that displays only the user-entered nofault settings.

System — A literal that displays only the netlist-described nofault settings.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example displays all pin names of the instances that have the nofault settings:

```
add_nofaults i_1006 i_1007 i_1008 -instance
report_nofaults
```

Related Commands

add_faults
add_faults
add_norace_clocks
delete_faults
delete_nofaults
report_faults

report_nonscan_cells

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_nonscan_cells [-All | -TIE0 | -TIE1 | -TIEX | -TLatch | -Clock_sequential | -INIT0 | -INIT1] [-VERilog] [{> | >>} *file_pathname*]

Description

Displays the non-scan cells whose model type you specify.

The **report_nonscan_cells** command displays the instance name, gate number, and model type of the non-scan cells that you specify. The tool assigns a model type to the non-scan cells during the Design Rules Check (DRC) in order to model non-scan memory behavior more exactly.

During scan loading, scan clocks can affect non-scan memory elements. The worst case is to assume that all non-scan memory elements will have an unknown state right after scan loading, which makes some faults difficult to test. Therefore, the tool assigns an appropriate model type to the non-scan cells, which sets them to known values. The following argument descriptions describe the conditions that the tool uses to make the various model type assignments.

Arguments

- -All

  An optional switch that displays all non-scan cells. This is the default.

- -TIE0

  An optional switch that displays the non-scan cells which are always 0 after each loading and before the next unloading.

  Non-scan cells that the tool models as TIE0 indicate that the pin constraints hold the cell's value during non-scan operation.

- -TIE1

  An optional switch that displays the non-scan cells which are always 1 after each loading and before the next unloading.

  Non-scan cells that the tool models as TIE1 indicate that the pin constraints hold the cell's value during non-scan operation.

- -TIEX

  An optional switch that displays the non-scan cells which are always unknown (X) after each loading and before the next unloading.

- -TLatch

  An optional switch that displays the non-scan cells which the tool models as transparent latches.

- -Clock_sequential

  An optional switch that displays all clock_sequential cells that are valid when the sequential depth is set to non-zero.

- -INIT0

  An optional switch that displays a list of cells which DRC has identified as being initialized to 0 by scan load but has failed to prove that it is tied to 0.

  Non-scan cells that the tool models as INIT0 indicate one of the following:

  - The test procedure can set the memory element to 0 after scan loading.
  - The non-off clock is a reset.

- -INIT1

  An optional switch that displays a list of cells which DRC has identified as being initialized to 1 by scan load, but has failed to prove that it is tied to 1.

  Non-scan cells that the tool models as INIT1 indicate one of the following:

  - The test procedure can set the memory element to 1 after scan loading.
  - The non-off clock is a set.

- -VERilog

  An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example reports only the non-scan cells that the tool models as transparent latches.

```
add_scan_groups g1 proc.g1
add_scan_chains c1 g1 scin scout
add_clocks 0 clk
set_system_mode analysis
report_nonscan_cells -tlatch
```

report_nonscan_models

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_nonscan_models [-Class {Full | System | User}]

Description

Displays the sequential non-scan model list.

Displays sequential models that you added by using the add_nonscan_models command, or that have the Dont_touch property in a Genie netlist.

Arguments

- -Class Full | System | User

  An optional switch and literal pair that specifies the source (or class) of the sequential non-scan models that you want to display. The valid literals are as follows:

  Full — A literal that specifies to display all the non-scan sequential models in the user and system class. This is the default.

  User — A literal that specifies to only display the non-scan sequential models that are the result of the add_nonscan_models command.

  System — A literal that specifies to only display the non-scan sequential models that are a result of the Genie netlist containing the Dont_touch property.

Examples

The following example displays all sequential non-scan models from the non-scan model list:

```
set_system_mode analysis
add_nonscan_models d_flip_flop1 d_flip_flop2
report_nonscan_models
```

Related Commands

add_nonscan_models

delete_nonscan_models

report_norace_clocks

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

Mode: setup, analysis

Usage

report_norace_clocks [{> | >>} *file_pathname*]

Description

Displays a list of all user-defined non-interacting clocks and their clock domain numbers.

The report_norace_clocks command displays a list of all clocks you specified as non-interacting using the add_norace_clocks command.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example creates two user-defined clock domains, each containing non-interacting clocks, then displays the current definitions.

set_system_mode analysis
add_norace_clocks clk1 clk2
add_norace_clocks clk3 clk4 clk5
report_norace_clocks

```
// User clock domain 1 : /clk1 /clk2
// User clock domain 2 : /clk3 /clk4 /clk5
```

Related Commands

add_clocks
add_norace_clocks
analyze_control_signals
delete_clocks
delete_norace_clocks

report_notest_points

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_notest_points [-Paths] [-tool_identified]

Description

Displays all the circuit points for which you do not want the tool to insert controllability and observability.

The report_notest_points command displays the circuit points added using the add_output_masks command and which therefore, the tool cannot use for testability insertion.

You can also list the critical path definitions that added notest points by using the -Paths switch.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

- -Paths

  An optional switch that displays the definitions for all currently loaded critical paths that have marked notest points. If this switch is not specified, the tool reports only the resulting notest points.

- -tool_identified

  An optional switch that displays only the notest points added by the tool.

  Note that if you issue the report_notest_points command without any switch, it will display the notest points added by the user.

Example

The following example displays the list of all circuit points that the tool cannot use for testability insertion:

```
add_notest_points i_1006/o i_1007/o
add_notest_points i_1009/o
report_notest_points
```

Related Commands

add_output_masks

delete_notest_points

report_output_masks

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

Mode: setup, analysis

Usage

report_output_masks [[{> | >>} *file_pathname*] [-Display {DEBug}]

Description

Displays a list of the currently masked primary output pins.

The report_output_masks command displays the masked primary output pins using the add_output_masks command. When you mask a primary output pin, the tool marks that pin as an invalid observation point during the fault detection process. The tool uses all unmasked primary output pins as possible observation points to which the effects of all faults propagate for detection.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

- -Display DEBug

  An optional switch and literal that specify to display the mask information for the primary output(s) as callouts in the Debug window.

Example

The following example masks two primary outputs and then displays the results:

add_output_masks qb1 qb2
report_output_masks

```
qb1
qb2
```

Related Commands

add_output_masks
delete_output_masks
set_output_masks

report_pattern_filtering

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_pattern_filtering [{> | >>} *file_pathname*]

Description

Displays the current pattern filtering configuration.

The report_pattern_filtering command displays the pattern filtering configuration set previously with the set_pattern_filtering command.

Note

The report_environment command reports if pattern filtering is on, but shows no status for pattern filtering when it is turned off.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example displays the current pattern filtering configuration:

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 7
   sampled_patterns_per_type     = 2
   sample_capture_clock          = on
   sample_pattern_type(s)        = all
-----------------------------------------------
```

Related Commands

report_environment
set_pattern_filtering
write_patterns

report_pattern_sets

Context: patterns ijtag

Mode: analysis

Usage

report_pattern_sets [-sort_by {declaration_order | name}] [*pattern_set_list*]

Description

Creates a report of a specified pattern set or of all pattern sets.

This command creates a human-readable report of all pattern sets when the pattern_set_list argument is omitted. The report contains the names of the pattern sets, the associated timeplates, the number of vectors, the TAP start and TAP end states, and whether the pattern set has been saved.

Arguments

- -sort_by {declaration_order | name}

  An optional switch and keyword pair that sorts the patterns in the report by order of declaration or by name. By default, the command sorts the patterns in the order of declaration. If you don't specify this switch, but do specify a pattern_set_list, the command reports pattern sets in the order specified by the list.

- *pattern_set_list*

  An optional Tcl list of pattern set names. If this list is present, the report is restricted to the pattern sets in the list. Otherwise, the report contains all pattern sets.

Return Value

None

Example 1

The following example creates a report of all pattern sets sorted by name:

```
report_pattern_sets -sort_by name
```

Example 2

The following example creates a report of the pattern sets returned by the lsearch command:

```
report_pattern_sets [lsearch -inline -all [get_pattern_set_list] p*]
```

For a detailed example, refer to "Creating Patterns Sets" in the *Tessent IJTAG User's Manual* .

Related Commands

close_pattern_set
get_open_pattern_set
get_pattern_set_list
open_pattern_set
reset_open_pattern_set

report_patterns

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

Mode: analysis

Usage

report_patterns [-external] [-pattern_index *pattern_index*]
[-path_faults] [-UNLoad_x] [{> | >>} *file_pathname*]

Available only when EDT is on:

report_patterns [-external] [-pattern_index *pattern_index*]
[-path_faults] [-edt [-chain_test] [-verbose] [-constant_chains]]
[-UNLoad_x] [{> | >>} *file_pathname*]

Description

Displays information about patterns.

You can display information for all patterns, or use the -Pattern_index switch to report on a single pattern. When you use the set_pattern_filtering command to create a temporary subset of sampled patterns, the report_patterns command reports only on patterns in the filtered subset. By default, information for internal patterns is reported, but you can report on patterns in an external pattern set by first specifying the external pattern file with the read_patterns command.

The following information is reported for each pattern when the -EDT switch is not used:

- Pattern # — Pattern index
- Pre-filtering pattern # — Pattern index in the original unfiltered pattern set (reported only when filtering by the set_pattern_filtering command is in effect, causing "Pattern #" to show pattern indexes in the filtered subset).
- Pattern type — Basic, Clock PO, Clock sequential, MacroTest, Multi-load, RAM sequential or Iddq
- Cycles — An integer indicating how many capture cycles there are in the pattern
- Loads — An integer indicating how many scan loads there are in the pattern
- Observe procedure — Name of the observe procedure (Master_Observe or Shadow_Observe), if applied in the pattern
- Capture procedure — Name of the named capture procedure, if applied in the pattern
- Capture clock sequence — Sequence the clocks are pulsed in during capture

When you use the -EDT switch to report a pattern created with Tessent TestKompress (EDT On), the information content of the report changes to display the following EDT-specific information:

- EDT block name if applicable

- EDT channel index number
- EDT channel names
- EDT input channel pin names
- EDT output channel pin names
- Observed core scan chain name(s)

If a channel is masked in the reported pattern, the name of the single scan chain observed for that channel is listed. If the channel is not masked in the pattern, this means that all connected chains are observed and the “<all connected chains>” string displays.

Arguments

- -external

  An optional switch that displays pattern information for external patterns instead of internal patterns. Before using this switch, you must specify the name of the external pattern file with the read_patterns command.

- -pattern_index *pattern_index*

  An optional switch and integer pair that displays information only for the pattern with the specified index, not all patterns.

- -path_faults

  An optional switch that displays the path faults detected by the internal patterns. This switch cannot be used with the -External switch and is only valid when the current fault type is set to path_delay.

- -edt **(Available only when EDT is on)**

  An optional switch that (1) reports which of the internal scan chains connected to each scan channel are observed by the specified EDT pattern when used in conjunction with the -Pattern_index switch, or (2) reports one line per pattern when used without the -Pattern_index switch. On each line, the pattern ID is followed by the pattern masking status for this pattern. This switch is valid only for Tessent TestKompress patterns created with Tessent TestKompress (EDT On).

- -chain_test **(Available only when EDT is on)**

  An optional switch that together with the -EDT and -Pattern_index switches reports on a particular chain test pattern in the pattern set. Without this switch, scan test pattern information is reported. Note that you can use this switch only when chain test patterns exist. Chain test patterns are created by a write_patterns command or if loaded as part of external patterns. If no chain test patterns exist, the tool issues an error message.

- -constant_chains **(Available only when EDT is on)**

  An optional switch that together with the -EDT and -Pattern_index switches lists the actual data source (decompressor or constant 0) for the specified internal pattern. Use this switch to display test pattern information for the low-power decompressor.

- -verbose **(Available only when EDT is on)**

  An optional switch that together with the -EDT and -Pattern_index switches lists the actual names of all the connected chains in the observed_chains column. Without this switch, the summary message "<all connected chains>" is displayed rather than the names of all the connected chains.

- -unload_x

  An optional switch that reports patterns that contain some scan cells with X unload values. For EDT patterns, X values due to EDT masking are not considered as unload X. When you specify this switch with the -pattern_index switch, the command also reports the scan cells that capture X values (see "Example 8" on page 1433).

  You can use the -unload_x switch for debugging purposes if you issue a "set_simulation_option -report_x_capturing_cells on" command and notice some patterns that capture X on scan cells. You should first use the -unload_x switch to see patterns that unload X, and then add the -pattern_index switch to further see which scan cells capture X of the specified pattern.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example of scan pattern reporting displays all patterns in the current internal pattern set. Table 7-30 summarizes the display notation.

Table 7-30. Notation Used in report_patterns Display

Notation	Column	Meaning
master_obs	observe_proc	Pattern applies a master_observe procedure.
shadow_obs	observe_proc	Pattern applies a shadow_observe procedure.
-	observe_proc	No master_observe or shadow_observe procedure is applied.
-	capture_proc	No named capture procedure is applied.
-	capture_clock_sequence	No clock is pulsed in the capture window.
[-]	capture_clock_sequence	No clock is pulsed in the capture window, but primary inputs are being forced.
[*clock1*][*clock2*]…	capture_clock_sequence	Sequence (left to right) in which *clock1* and *clock2*, etc. are pulsed.

Table 7-30. Notation Used in report_patterns Display

Notation	Column	Meaning
[*clock1*, *clock2*, …]	capture_clock_sequence	Shows *clock1* and *clock2*, etc. are pulsed at the same time.
S	capture_clock_sequence	A scan load occurs in the corresponding cycle.[1]

1. The first cycle always has a scan load, so no S is displayed for the first cycle.

> **report_patterns**

```
patt #  type            cycles loads observe_proc cap_proc    cap_clk_seq
------- --------------- ------ ----- ------------ ----------- -----------
   0    basic             1      1   master_obs     -         [clk1]
   1    basic             1      1     -            -         [clk1]
   2    clock_po          1      1     -            -            -
   3    clock_seq         2      1   shadow_obs   named_cap1  [clk1]
                                                               [clk1]
   4    multi_load        3      3     -            -         [ramclk]
                                                              S[clk1]
                                                              S[clk2]
   5    clock_seq         2      1     -            -         [clk1,clk2]
                                                              [-]
   6    clock_seq         2      1     -            -         [clk1][-]
   7    multi_load        3      2     -            -         [clk1]
                                                              S[clk1]
                                                              [clk1]
   8    basic             1      1     -            -         [clk1]
   9    basic             1      1     -            -         [-]
  10    basic             1      1     -            -         [scn_rst]
```

Pattern 0 is a basic pattern that applies a master_observe procedure. Pattern 2 is a clock PO pattern; therefore, no clock is pulsed in the capture window. Pattern 3 is a clock sequential pattern that uses the named capture procedure “nmd_cap1” in which the clock sequence is [clk1] [clk1]. Pattern 4 is a multiple-load pattern. The S in the capture clock sequence indicates there is a scan load in the corresponding cycle (the first cycle always has a scan load, so no S is displayed there). Pattern 5 is a clock sequential pattern that pulses two clocks simultaneously in the first capture cycle, then in the cycle that follows, forces primary inputs without pulsing any clocks. Pattern 6 differs from pattern 5 in that it pulses just one clock in the first capture cycle.

Example 2

The following example shows how you can use the set_pattern_filtering command to display a subset of patterns. The example first displays just the non-basic patterns of the preceding example, then displays just the patterns that use ramclk as a capture clock:

set_pattern_filtering -range 2 7

```
//  Internal pattern filtering is on: 6 selected pattern(s), 11 original
    pattern(s).
```

report_patterns

```
       pre-
       filter
```

```
patt # patt # type        cycles loads observe_proc cap_proc   cap_clk_s
------ ------ ----------- ------ ----- ------------ ----------- ---------
   0      2   clock_po      1      1     -            -            -
   1      3   clock_seq     2      1   shadow_obs   named_cap1  [clk1]
                                                                 [clk1]
   2      4   multi_load    3      3     -            -         [ramclk]
                                                                S[clk1]
                                                                S[clk2]
   3      5   clock_seq     2      1     -            -       [clk1,clk2]
                                                               [-]
   4      6   clock_seq     2      1     -            -         [clk1][-]
   5      7   multi_load    3      2     -            -         [clk1]
                                                                S[clk1]
                                                                [clk1]
```

set_pattern_filtering -clocks ramclk

```
//  Warning: The previous filter options have been reset.
//  Internal pattern filtering is on: 1 selected pattern(s), 11 original
    pattern(s).
```

report_patterns

```
       pre-
       filter
patt # patt # type        cycles loads observe_proc cap_proc   cap_clk_s
------ ------ ----------- ------ ----- ------------ ----------- ---------
     0      4 multi_load    3      3     -            -        [ramclk]
                                                               S[clk1]
                                                               S[clk2]
```

Example 3

The following EDT pattern reporting example reports EDT chain test pattern 9. The design is a Modular Compressed ATPG design with two EDT blocks, p1 and p2. Block p1 has one channel and all chains connected to this channel are observed with this pattern. Block p2 has three channels. For each of them, only one chain is observed with this pattern.

report_patterns -pattern_index 9 -edt -chain_test

```
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// ----- --- ------------ ----------------- ------------------ ----------
// p1     1  p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1  <all conn.
//                                                             chains>
// ---------------------------------------------------------------------
// p2     1  p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1 core2_chain1
// "      2  p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2 core2_chain4
// "      3  p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3 core2_chain7
```

Example 4

The following example is the same as the preceding example except that the -Verbose switch is included in the command. This expands the “<all connected chains>” summary information to the names of all chains connected to p1_channel1.

report_patterns -pattern_index 9 -edt -chain_test -verbose

```
// block ch# channel_name channel_input_pin channel_output_pin obs_chains
// ----- --- ------------ ----------------- ------------------ ----------
```

```
// p1     1  p1_channel1  core1_edt_ch_in1  core1_edt_ch_out1 core1_chain1
//                                                           core1_chain2
//                                                           core1_chain3
//                                                           core1_chain4
//                                                           core1_chain5
// ---------------------------------------------------------------------
// p2     1  p2_channel1  core2_edt_ch_in1  core2_edt_ch_out1 core2_chain1
// "      2  p2_channel2  core2_edt_ch_in2  core2_edt_ch_out2 core2_chain4
// "      3  p2_channel3  core2_edt_ch_in3  core2_edt_ch_out3 core2_chain7
```

Example 5

The following example reports on a particular pattern in an external pattern set for a design that has both uncompressed scan chains (direct access chains) and compressed scan chains. The direct access chains are reported in lines that begin “<uncompressed chain>”. As the design consists of just one block, the “block” column is unnecessary and it is not displayed.

read_patterns my_edt_patts
report_patterns -external -pattern_index 199 -edt -chain_test

```
// ch# channel_name channel_input_pin channel_output_pin obs_chains
// --- ------------ ----------------- ------------------ ------------
//  1  pic_channel1 core1_edt_ch_in1  core1_edt_ch_out1  core1_chain1
//  2  pic_channel2 core1_edt_ch_in2  core1_edt_ch_out2  core1_chain4
//  3  pic_channel3 core1_edt_ch_in3  core1_edt_ch_out3  core1_chain7
// ---------------------------------------------------------------------
// <uncompressed chain> core2_edt_ch_in1  core2_edt_ch_out1  core2_chain1
// <uncompressed chain> core2_edt_ch_in2  core2_edt_ch_out2  core2_chain2
```

Example 6

The following example sets the fault type to path_delay, loads path data into the system, and creates patterns to test the path delay faults. It uses the report_patterns command to check which path faults are detected by the internal patterns.

set_fault_type path
read_fault_sites paths.txt
set_pattern_type -sequential 2
create_patterns
report_patterns -path

```
//  Pattern 0 detects following path fault(s):
//    Detected path   = path1, edge = rise, detection = robust
//    Detected path   = path44, edge = fall, detection = robust
//  Pattern 1 detects following path fault(s):
//    Detected path   = path3, edge = rise, detection = robust
//    Detected path   = path12, edge = fall, detection = robust
//    Detected path   = path16, edge = fall, detection = robust
//    Detected path   = path21, edge = fall, detection = robust
//    Detected path   = path33, edge = fall, detection = robust
//    Detected path   = path34, edge = fall, detection = robust
//    Detected path   = path45, edge = fall, detection = robust
//    Detected path   = path52, edge = fall, detection = robust
//  Pattern 2 detects following path fault(s):
//    Detected path   = path5, edge = fall, detection = robust
//    Detected path   = path44, edge = rise, detection = robust
...
```

Example 7

The following example shows the data source for scan chains associated with the 101 test pattern index.

report_patterns -edt -constant_chains -pattern_index 101

```
//
//   block                data source statistics
//  -------  ------------------------------------------------
//  piccpu1  8 of 8 scan chains are driven by constant value
//  -------------------------------------------------------
//  piccpu2  3 of 8 scan chains are driven by constant value
//  -------------------------------------------------------
//  piccpu3  8 of 8 scan chains are driven by constant value
//
//  command: rep patterns  -edt -constant_chains  -patt 101 -verbose
//
//   block       chain      data source
//  -------  ------------  ------------
//  piccpu1  core1_chain1  constant 0
//     "     core1_chain2  constant 0
//     "     core1_chain3  constant 0
//     "     core1_chain4  constant 0
//     "     core1_chain5  constant 0
//     "     core1_chain6  constant 0
//     "     core1_chain7  constant 0
//     "     core1_chain8  constant 0
//  -----------------------------------
//  piccpu2  core2_chain1  decompressor
//     "     core2_chain2  decompressor
//     "     core2_chain3  constant 0
//     "     core2_chain4  constant 0
//     "     core2_chain5  decompressor
//     "     core2_chain6  constant 0
//     "     core2_chain7  decompressor
//     "     core2_chain8  decompressor
//  -----------------------------------
//  piccpu3  core3_chain1  constant 0
//     "     core3_chain2  constant 0
//     "     core3_chain3  constant 0
//     "     core3_chain4  constant 0
//     "     core3_chain5  constant 0
//     "     core3_chain6  constant 0
//     "     core3_chain7  constant 0
//     "     core3_chain8  constant 0
//
```

Example 8

The following example shows how to report patterns that contain some scan cells with X unload values:

report_patterns -unload_x

```
//  pattern #   type  cycles  loads  unload_X observe_proc  capture_proc  capture_clock_seq
//  ---------  -----  ------  -----  -------- ------------  ------------  -------------------
//        0    basic     1      1      yes        -             -        [clock]
//        1    basic     1      1      yes        -             -        [clock]
//        4    basic     1      1      yes        -             -        [clock]
```

```
//        5    basic     1      1      yes        -             -        [clock]
//        7    basic     1      1      yes        -             -        [clock]
```

You can then further investigate one of these patterns and report the scan cells that that capture X values:

report_patterns -pattern_index 1 -unload_x

```
// pattern # type  cycles loads unload_X observe_proc capture_proc capture_clock_seq
// --------- ----- ------ ----- -------- ------------ ------------ -------------------
//       0   basic    1     1     yes       -             -        [clock]
// Scan cell dff1 (233) captures X
// Scan cell dff3 (235) captures X
```

Related Commands

read_patterns
set_pattern_filtering
set_simulation_options

report_power_data

Context: all contexts

Mode: all modes

Usage

report_power_data {[-Summary | -All | -DOmains [*domain_name* …] | -STates [*state_name* …]} [-ISolation_cells] [-LEvel_shifters] [-Always_on_cells] [-REtention_cells]

Description

Reports the power data currently loaded in the system from either a read_cpf or read_upf command.

Arguments

- -Summary

  An optional switch that specifies the content of power data to be reported. By default, the command reports the summary of loaded power data.

- -All

  An optional switch that reports the details of the loaded power data, including all power domains and all power modes. When executing the command in non-setup mode, the tool reports the count of low power cells associated with each power domain and the gate count of every power domains.

- -DOmains [*domain_name* …]

  An optional switch and optional repeatable string that reports only the specified power domains (*domain_name*) and the data associated with the specified power domains. If no power domain is specified, all power domains are reported.

- -STates [*state_name* …]

  An optional switch and optional repeatable string that reports only the specified power mode and power data related to the specified power modes. If no power mode is specified, all power modes will be reported.

- -ISolation_cells

  An optional switch that reports the detailed information of the isolation cells.

- -LEvel_shifters

  An optional switch that reports the detailed information of the level shifters.

- -Always_on_cells

  An optional switch that reports the detailed information of the always-on cells.

- -REtention_cells

  An optional switch that reports the detailed information of the retention cells.

Example 1

The following example displays detailed power data for isolation cells:

report_power_data -isolation_cells

Related Commands

add_faults
delete_faults
read_faults
read_cpf
report_faults
report_power_data

report_power_metrics

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_power_metrics [-PATterns {**ALL** | ***pattern_index*…** | **PAD**
| **RANGE** ***first_pattern_index last_pattern_index***}
[{-SORT_Capture [ST_Peak | ST_Average | WSA_Peak | WSA_Average]
| -SORT_Shift [LOAD | RESPonse]} [-ASCending | -DEScending]]
[-INTernal | -EXTernal] [-EVERY_CAPture_cycle]
[-INSTance ***instance_pathname***...] [-MODUle ***module_name***...]
[-SHIFT [-SCan_test | -CHain_test | -ALl_test]
[-CYCLE {**ALL** | **PEAK** | ***cycle_index***... | RANGE ***first_cycle_index last_cycle_index***}]]
[{> | >>} ***file_pathname***]

Description

Displays shift and capture power metrics for specified test patterns.

Use this information to identify the test patterns that may cause chip failure due to power issues. Enter this command without any arguments to display a summary of shift and capture power metrics for all test patterns.

Note

You can use this command when the set_power_metrics command is set to -Shift off -Capture off and the test patterns are automatically resimulated to calculate shift and capture power if necessary.

Arguments

- -PATterns {ALL | *pattern_index*… | PAD | RANGE ***first_pattern_index last_pattern_index***…}

  An optional switch and literal, list of positive integers, or literal and list of positive integers that display the shift power and capture power for specified test patterns. Options include:

  ALL — Displays the shift power and capture power for all patterns.

  pattern_index — Displays the shift power and capture power for the specified patterns. Pattern index starts from 0.

  PAD — Displays the shift power consumed by the unloading of the last test pattern in the specified test patterns. This value is only available when set_power_metrics -composite_shift_power is enabled.

  RANGE ***first_pattern_index last_pattern_index*** — Displays the shift power and capture power for all patterns from *first_pattern_index* to *last_pattern_index*.

- -SORT_Capture [ST_Peak | ST_Average | WSA_Peak | WSA_Average]

  An optional switch and literal pair that sorts specified test patterns based on their switching activity during capture. Literal options include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  W**SA_Average** — Average of weighted switching activity.

  For more information, on WSA and ST values, see Example 1 below and the set_power_metrics command.

- -SORT_Shift [LOAD | RESPonse]

  An optional switch and literal pair that sorts the specified test patterns based on their function during test pattern application. This option i only valid when the -SHIFT option is specified. Literal options include:

  LOAD — Switching activity during the loading of test pattern stimuli.

  RESPonse — Switching activity during the unloading of test responses.

- -ASCending | -DEScending

  An optional switch that sorts the specified test patterns in an ascending or descending order based on switching activity. By default, test patterns with the lowest switching activity are listed first (ascending order).

- -INTernal | -EXTernal

  An optional switch that specifies whether power metrics are reported for the internal or external test pattern set. By default, the internal test pattern set is used.

- -EVERY_CAPture_cycle

  An optional switch that reports the weighted switching activity and the “state element transition” for every capture cycle on a per pattern basis. By default, the tool reports only the average and peak capture activity for the test pattern.

- -INSTance ***instance_pathname***...

  An optional switch and repeatable string that reports the shift and capture power usage for one or more specified design instances. The instance_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -MODULe ***module_name***...

  An optional switch and repeatable string that reports the shift and capture power usage for all instances of a specified design module(s).

- [-SHIFT {-SCan_test | -CHain_test | -ALl_test}]

  An optional pair of switches that reports shift and capture power metrics for a specific pattern type. By default, the power metrics for scan test patterns are reported. Options include:

-SCan_test — Scan test patterns. Default.

-CHain_test — Chain test patterns.

-ALL_test — Scan and chain test patterns.

- [-CYCLE {**ALL** | **PEAK** | ***cycle_index*** ... | RANGE ***first_cycle_index last_cycle_index***}]

  An optional switch and literal, list of integers, or a literal and a pair of integers that reports shift power for the specified test pattern cycles. You must enable the composite shift power calculation with set_power_metrics to calculate these values. Options include:

  ALL — Shift transitions for all cycles.

  PEAK — Shift transitions for the cycle with the peak shift power.

  cycle_index — Shift transitions for the specified cycles.

  RANGE *first_cycle_index last_cycle_index* — Shift transitions for a specified range of cycles.

- > ***file_pathname***

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> ***file_pathname***

  An optional redirection operator and pathname pair for appending the contents of *file_pathname*.

Example 1

The following example displays the shift power and capture power for pattern 15.

report_power_metrics -patterns 15

```
//                  Shift                          Capture
//  Pattern    Load    Response          WSA          State Element Tran.
//  -------  --------  --------  -------------------  -------------------
//  15       45.29%   40.56%    13.89% (15.49%)     14.32% (18.13%)
```

- The Shift Load number (45.29%) is the percentage of the number of scan cell transitions during the loading of the test stimuli over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.

- The Shift Response number (40.59%) is the percentage of the number of scan cell transitions during the unloading of the test response over the worst case. The worst case is calculated by assuming that every scan cell is toggled in every shift cycle.

- The first Capture WSA (Weighted Switching Activity) number (13.89%) is the average of the weighted switching activity per capture cycle. The second number (15.49%) is the weighted switching activity for the peak capture cycle. The peak capture cycle is the cycle with the highest weighted switching activity among all capture cycles in a test pattern.

The WSA calculation includes the switching activity of the combinational logic, state elements, as well as factoring in gate output fanout for capture cycles.

- The first Capture State Element Tran. number (14.32%) is the average number of state element transitions per capture cycle. The second number (18.13%) is the number of state element transitions in the peak capture cycle. The peak capture cycle is the cycle with the highest number of state element transitions among all capture cycles in a test pattern.

Example 2

The following example displays a summary of capture power and shift power for all patterns:

report_power_metrics

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                          9.03%   18.74%  36.82%
State Element Transitions    8.99%   25.44%  47.33%
--------------------------  ------  -------  ------
Load Shift Transitions      39.06%   44.62%  51.66%
Response Shift Transitions   6.45%   39.38%  50.54%
```

Example 3

The following example calculates and reports the power metrics for external test patterns, *mypat.ascii*:

read_patterns mypat.ascii
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
```

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests for all test patterns in the internal test pattern set:

set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all

```
// Chain test :
//
//           Pattern          Shift
//  Pat. loaded Pat. unloadedAverage
```

```
//  ----------- -------------   -------
//  0           -               32.89%
//  1           0               48.60%
//  2           1              54.20%
//  pad         2              49.56%
//
// Scan test :
//
//           Pattern          Shift
//  Pat. loaded Pat. unloadedAverage
//  ----------- -------------   -------
//  0            -              10.60%
//  1            0               8.60%
//  2            1              14.20%
//  3            2               8.20%
//  pad          3               9.56%
```

Example 5

The following example shows the composite shift power for a specified range of cycles in the second and third test pattern in the internal test pattern set. In this example, the peak shift power for test pattern 1 occurs in cycle 6 as indicated by the asterisk cycle index:

set_power_metrics -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle range 4 6

```
//          Pattern           Cycle  | Shift Transitions
// Pat. loaded Pat. unloaded
// ----------- -------------  ------ | -----------------
// 1            0              4     |            43.82%
//                             5     |            40.76%
//                             6  (*)|            47.64%
// 2            1              4     |            38.82%
//                             5     |            39.54%
//                             6     |            37.56%
//
// (*) indicates the cycle with the peak shift power.
```

Example 6

The following example reports the peak shift power for test pattern 1 and 2:

set power metric -composite_shift on
report_power_metrics -pattern 1 2 -shift -cycle peak

```
//          Pattern        Peak Cycle | Shift Transitions
// Pat. loaded Pat. unloaded
// ----------- ------------- -----------| -----------------
// 1            0             5         |            40.76%
// 2            1             114       |            42.82%
```

Example 7

The following example shows the switching activity for capture when the -every_capture_cycle argument is specified. In this example, there are three capture cycles in pattern 10.

```
ATPG> report power metrics -pattern 10 -every_capture_cycle
//  Pattern                     Capture
//                   WSA            State Element Tran.
//  -------  ---------------------  ---------------------
//  10       8.23%, 25.21%, 24.68%  1.69%, 22.16%, 19.53%
```

The following example has patterns with different numbers of capture cycles. This occurs when NCPs or clock control definitions are used. This example shows patterns with 1, 2, and 3 capture cycles.

```
ATPG> report power metrics -pattern range 12 15 -every_capture_cycle
//  Pattern                     Capture
//                   WSA            State Element Tran.
//  -------  ---------------------  ---------------------
//  12       8.16%, 21.91%, 23.22%  1.54%, 16.64%, 19.08%
//  13       8.22%, 25.06%,         1.74%, 21.28%
//  14       8.17%, 21.47%, 23.40%  1.52%, 15.78%, 14.42%
//  15       8.21%,                 1.61%,
```

Related Command

set_power_metrics

report_primary_inputs

Context: all contexts

Mode: all modes

Usage

report_primary_inputs [-All | *net_pathname*... | *primary_input_pin*...] [-Class {Full | User | System}] [-Verbose] [{> | >>} *file_pathname*]

Description

Displays a list of the primary inputs of a circuit.

You can choose to display either the user class, system class, or full classes of primary inputs. Additionally, you can display all of the primary inputs or a specific list of primary inputs. If you issue the command without specifying any arguments, then the tool displays all the primary inputs.

Arguments

- -All

  An optional switch that displays all the primary inputs. This is the default.

- *net_pathname*

  An optional repeatable string that specifies the circuit connections whose user-class primary inputs to display.

- *primary_input_pin*

  An optional repeatable string that specifies a list of system-class primary input pins to display.

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the primary input pins to display. The literal choices are as follows:

  Full — Displays all the primary input pins in the user and system class. This is the default.

  User — Displays only the user-entered primary input pins.

  System — Displays only the netlist-described primary input pins.

- -Verbose

  An optional switch that displays all the nets grouped under a primary input.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for creating or replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list for appending to the contents of *file_pathname*.

Example 1

The following example displays the full classes of primary inputs:

add_primary_inputs indata2 indata4
report_primary_inputs -class full

Example 2

The following example displays the user classes of primary inputs:

add_primary_inputs -internal /dut/*/CLK_X -pin_name MYCLKX

```
//Note: Adding primary input MYCLKX merging 224 nets.
```

report_primary_inputs -class user

```
USER: /MYCLKX (merged internal pin)
Merges 224 nets
```

Example 3

The following dft -scan example displays all of the primary inputs:

report_primary_inputs -all

```
SYSTEM:    /clk
SYSTEM:    /datain
```

Note

The label “system” means that these are primary inputs that the tool automatically recognizes because they were in the netlist. Because there is no add_primary_inputs command in the tool as there is in Tessent FastScan, all primary inputs are of the system-defined class.

Related Commands

add_primary_inputs
delete_primary_inputs
report_primary_outputs

report_primary_outputs

Context: all contexts

Mode: all modes

Usage

report_primary_outputs [-All | *net_pathname*... | *primary_output_pin*...]
[-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

Displays the specified primary outputs.

The report_primary_outputs command displays a list of primary outputs of a circuit. You can choose to display either the user class, system class, or full classes of primary outputs. Additionally, you can display all of the primary outputs or a specific list of primary outputs. If you issue the command without specifying any arguments, then the tool displays all the primary outputs.

Arguments

- -All

  An optional switch that displays all primary outputs. This is the default.

- *net_pathname*

  An optional, repeatable string that specifies the circuit connections whose user-class primary outputs you want to display.

- *primary_output_pin*

  An optional, repeatable string that specifies a list of system-class primary output pins that you want to display.

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the primary output pins that you want to display. The literal choices are as follows:

  Full — A literal that displays all of the primary output pins in the user and system class. This is the default.

  User — A literal that displays only the user-entered primary output pins.

  System — A literal that displays only the netlist-described primary output pins.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example displays all primary outputs in the user class:

add_primary_outputs outdata1 outdata3 outdata5
report_primary_outputs -class user

Example 2

The following dft -scan example displays all of the primary outputs:

report_primary_outputs -all

```
SYSTEM:     /dataout
SYSTEM:     /dataout1
```

Note

The label “system” means that these are primary outputs that the tool automatically recognizes because they were in the netlist. Because there is no add_primary_outputs command in the tool as there is in Tessent FastScan, all primary outputs are of the system-defined class.

Related Commands

add_primary_outputs
delete_primary_outputs
report_primary_inputs

report_procedures

Context: dft -edt, dft -scan, patterns -scan, patterns -ijtag, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Usage

report_procedures {***procedure_name*** [*group_name*] [-Display Wave] | -**All**} [{> | >>} *file_pathname*]

Description

Displays the specified procedure.

The report_procedures command displays all procedures or the specified procedure.

Arguments

- ***procedure_name***

  A string that specifies which procedure to display.

- *group_name*

  An optional string that specifies a particular scan group from which to display the specified procedure.

- -Display Wave

  An optional switch that displays the specified procedure in the Wave window of DFTVisualizer. This is for unoptimized NCP's. Use the report_capture_procedures command to display optimized NCP's.

- -**All**

  A switch that specifies for the tool to display all procedures. This is the default.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Example

The following example displays all procedures:

```
report_procedures -all
```

Related Commands

add_scan_groups
read_procfile
report_capture_procedures
report_timeplate
write_patterns
write_procfile

report_processors

Context: unspecified, all contexts

Mode: setup, analysis, insertion (dft -edt context only)

Usage

report_processors [-Verbose]

Description

Displays information about the processors used for the current multiprocessing environment.

For more information, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

The report_processors command displays the information listed in Table 7-31 for the current multiprocessing environment:

Table 7-31. Default report_processors Information Display

Column Name	Description
hosts/processors	Network names of host machines, along with an identification number for each slave process
arch	Architecture of host machines (similar to entering "uname -m" on each host)
CPU(s)	Number of hardware processors and their speed for each host machine
%idle	Current CPU idle percentage (a small interval of time is sampled)
free RAM	A machine's total free memory
process size	Size of the master or slave process

This information enables you to determine whether to use the current set of host machines for slave processes or whether to use different machines. For example, a host machine with a low %idle is not as likely to provide as much CPU time to slave processes as one that is 100% idle. The free RAM may also help you decide which machines to avoid at the present time.

Note

Be sure to consider the number of processors on the host machine when evaluating the idle percentage (%idle), as a partially idle machine may have one or more completely idle processors available to run slave processes.

Arguments

- -Verbose

An optional switch that adds the data listed in Table 7-32 to the normal display:

Table 7-32. Additional Data Displayed by -Verbose

Column Name	Description
free swap	Total available swap space minus the swap space currently in use
total RAM	Total available RAM
total swap	Total available swap space
user CPU	Current sum of the time spent executing the master or slave process
sys CPU	Current sum of the time spent in system routines on behalf of the master or slave process

Example

The following example summarizes the performance statistics for the current multiprocessing environment:

report_processors

```
hosts/processors   arch     CPU(s)      %idle    free RAM    process size
----------------  ------  -----------  -----  ------------  ------------
nemo (master)     x86-64  2 x 2.2 GHz    99%       5.82 GB      40.82 GB
ahab          1   x86-64  2 x 2.0 GHz     0%      10.21 GB      20.45 GB
              2                                               20.45 GB
```

Related Commands

add_processors
delete_processors

report_read_controls

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

Mode: setup, analysis

Usage

report_read_controls [{> | >>} *file_pathname*]

Description

Displays all of the currently defined read control lines.

The report_read_controls command displays all the read control lines specified using the add_read_controls command. The display also includes the corresponding off-state with each read control line.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example

The following example displays a list of the current read control lines:

```
add_read_controls 0 r1 r3
add_read_controls 1 r2 r4
report_read_controls
```

Related Commands

add_read_controls
delete_read_controls
analyze_control_signals

report_register_value

Context: all contexts

Mode: all modes

Usage

report_register_value [(> | >>) *file_pathname*]

Description

Reports register information for each register name.

In the report, the columns contain the following data:

- Register Value Name — Name of the register specified by the add_register_value command in the pattern generation dofile.
- Value string — Binary value loaded into the register, if known. The register is loaded when the write_patterns command is executed. Before write_patterns is executed, this column contain Xs for registers with dynamic contents. For register values with a width greater than 200, only the first 200 bits of the value are displayed.
- Width — Number of bits in the register.
- Dynamic/Static — Shows the DRC impact.
  - Dynamic: The contents of the register is not known/not used during DRC and can be changed.
  - Static: The contents of the register is known and used during DRC and cannot be changed.
- Source — The source of the register contents: user-defined or tool-determined.
- LSB/MSB First — Whether the register is applied from MSB > LSB or LSB > MSB.
- Pin Inversion — Indicates whether any inversions are present. Contains two values that refer to input_pin_inversion and output_pin_inversion: F = false and T = true.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example reports the values of LPCT configuration data fields before and after the write_patterns command is executed. In this example, if you issue "set_lpct_controller -max_shift 1000", the lpct_config_shift_length register has the width of 10, as shown below.

report_register_value

Report output before the write_patterns command is executed:

```
Register value name            Value    Width Dynamic/ Source    LSB/MSB Pin
                               string        Static             First   inver-
                                                                       sion
-------------------            ------   ----- -------- ------   ------- ------
lpct_config_capture_depth      xx       2     dynamic  capture  LSB     FF
                                                       width    first

lpct_config_chain_pattern_     XXXXXXX  10    dynamic  chain    LSB     FF
load_count                     XXX                    patterns load/ first
                                                       load/
                                                       unload
                                                       count
lpct_config_edt_bypass         0        1     static   user     MSB     FF
                                                       specifie first
                                                       d
lpct_config_reset_control      0        1     static   user     MSB     FF
                                                       specifie first
                                                       d
lpct_config_scan_en_control    0        1     static   user     MSB     FF
                                                       specifie first
                                                       d
lpct_config_scan_pattern_      XXXXXXX  20    dynamic  scan     LSB     FF
load_count                     XXXXXXX                patterns first
                               XXXXxx                 load/
                                                      unload
                                                      count
lpct_config_shift_length       XXXXXXX  10    dynamic  shift    LSB     FF
                               XXX                    length   first
```

Report output after the write_patterns command is executed:

```
Register value name         Value     Width  Dynamic/  Source     LSB/MSB  Pin
                            string           Static                First    inver-
                                                                          sion
-------------------         ------    -----  --------  ------     -------  ------
lpct_config_capture_depth   10        2      dynamic   capture    LSB      FF
                                                       width      first

lpct_config_chain_          00000101  10     dynamic   chain      LSB      FF
pattern_load_count          10                         patterns   first
                                                       load/
                                                       unload
                                                       count
lpct_config_edt_bypass      0         1      static    user       MSB      FF
                                                       specified  first
lpct_config_reset_control   0         1      static    user       MSB      FF
                                                       specified  first

lpct_config_scan_en_        0         1      static    user       MSB      FF
control                                                specified  first

lpct_config_scan_pattern_   00000000  20     dynamic   scan       LSB      FF
load_count                  00010111                   patterns   first
                            1111                       load/
                                                       unload
                                                       count
lpct_config_shift_length    00001011  10     dynamic   shift      LSB      FF
                            01                         length     first
```

Related Commands

add_register_value
delete_register_value
set_register_value

report_resources

Context: unspecified, all contexts

Mode: all modes

Usage

report_resources

Description

Displays the machine resources used by the active tool session.

The report_resources command displays the cumulative processing time and the process memory usage for the machine on which the tool is running.

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

Example

The following example reports on machine resources:

report_resources

```
// Machine Name: bongo
// User Name: smith
// User CPU Time: 27.8 seconds
// System CPU Time: 1.6 seconds
// Memory Used: 141.739 MB
```

report_scan_cells

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

Mode: setup, analysis, insertion (dft -scan context only)

Usage

report_scan_cells [-All | {*chain_name* [{-Range *cell_id1 cell_id2*} | *cell_id*…]}...]
[-VERilog] [-Power_domain]
[-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]
[{> | >>} *file_pathname*]

"dft -scan" Usage

report_scan_cells [-All | *chain_name*…] [-SHift_register_flops]
[-Filename *filename* [-Replace]]
[-Display {[DEBug] [DESign] [DAta] | ALl} [APpend]]
[-Power_domain]
[{> | >>} *file_pathname*]

Description

Displays a report on the scan cells that reside in the specified scan chains.

The report_scan_cells command provides a report on the scan cells within specific scan chains. The report provides the following information for each scan cell:

- Chain cell index number (cell ID), where 0 is the scan cell closest to the scan-out pin.
- Scan chain in which the scan cell resides.
- Scan group in which the scan cell resides. (The scan group data will be hidden if the design has only scan group.)
- Sequential element type, primitive type, and latch triggering type during shifting. The latch triggering type can be one of the following: LE, TE, AH, AL.
- Inversion data for each sequential element.
- Gate index number for each sequential element.
- Shift clock name and inversion status at the clock input of each sequential element. ("dft -edt" and "patterns -scan" only)
- Top-level library model cell name of each sequential element.
- Instance name of each sequential element.
- Library instance name, if one is defined.
- Cell input and output pins.

For more information, see "Scan Terminology" in the *Tessent Scan and ATPG User's Manual*.

If you issue the command without specifying any arguments, then the tool displays a report on the scan cells for all scan chains.

Tip: If you are saving a log of the tool session, reporting all scan cells will fill the logfile with information for every scan cell—a lot of information if the design is large! To reduce the impact on the logfile in this case, consider reporting on only one scan chain.

Note

Although scan cells are listed in the order of nearest-to-output first, the sequential elements inside one cell are not listed in that order (the master is always listed first).

Be aware that when reading hierarchical names into its internal database, the tool hides backslashes (\), converts periods (.) to slashes (/), and ignores spaces. For example, the tool reads the cell name, “FFH.\E/MY_SCAN_FF .SCAN_FF5” in a WGL pattern, as “FFH/E/MY_SCAN_FF/SCAN_FF5”, storing the latter in its internal database. To report on this cell, you must reference it by the name that is stored internally.

Note

The difference between the internal hierarchical name and the name in the WGL file of the preceding example does not alter the correctness of the tool. That is, the output pattern file remains consistent with the original netlist.

dft -scan only: If you issue the command with the -Filename switch, then the tool writes the scan cells to a file in the format that can be read by the insert_test_logic command. The format of the written file is different than the format of the viewed report. You can optionally edit the scan cell order in the file before reading the file with the insert_test_logic command.

Arguments

- -All

  An optional switch that displays information for all the scan cells in all the scan chains. This is the default.

- *chain_name*

  An optional, repeatable string that specifies a scan chain whose scan cell data you want to display. When you use this argument without accompanying cell IDs, the tool displays information for all the scan cells in the scan chain. If you follow the *chain_name* argument with the -Range switch (and beginning and ending cell IDs for the range), the tool displays information for every cell in the range. If you provide one or more cell IDs without the -Range switch, the tool displays information for just those cells in that chain.

- -Range *cell_id1 cell_id2*

  An optional switch and pair of integers that specify the beginning and ending cell IDs of a range of scan cells in *chain_name* whose data you want to display.

- *cell_id*

  An optional repeatable integer that specifies the index number (cell ID) of a scan cell and, together with the *chain_name* argument, identifies a specific scan cell whose data you want to display.

- -VERilog

  An optional switch that outputs the cell instance and pin pathnames in Verilog syntax, rather than using the default netlist independent format.

- -SHift_register_flops (dft -scan only)

  An optional switch that specifies to print only the shift register flip-flops that are stitched into scan chains.

- -Filename *filename* [-Replace] (dft -scan only)
- An optional switch and string that specifies that the tool writes the list of scan cells to a file. The format of the written file is different than the format of the viewed report; see the insert_test_logic filename argument for detailed information on the file format. The -*Replace* switch specifies that the file should be overwritten if it already exists.

  -Display {[DEBug] [DESign] [DAta] | ALl} [APpend]

  An optional switch, repeatable literal, and literal that display the reported information graphically in the specified DFTVisualizer window(s); if *Append* is specified, the information is added to the existing contents of the window instead of replacing the contents. The valid options for the *window_name* argument, from which you can select as many as you want, are as follows:

  DEBug — Debug window

  DESign — Design window

  DAta — Data window

  ALl — A literal that displays the information in all of the preceding windows.

- -Power_domain

  An optional switch that creates a “Power Domain” column in the report when applicable.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

This example displays a list of the scan cells for scan chain0. The design only has one scan group; therefore, the scan group column is not shown.

report_scan_cells chain0

```
cell# chain   memory_type    inv gate# shift_clock inv cell_name instance_name (ext.pin
names)
----- ------ -------------- ---- ----- --------------- --------- ------------- ----------
0     chain0 MASTER (LA-AH) FFTT 24848 /clk        F   DFF3      /Q_I4  ""     (TI,QN)
1     chain0 MASTER (LA-AH) FFFF 25040 /clk        F   DFF1      /H_14  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24849 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
2     chain0 MASTER (LA-AH) FFFF 24546 /clk        F   DFF1      /R_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 25041 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
3     chain0 MASTER (LA-AH) FFFF 24542 /clk        F   DFF1      /R_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24547 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
4     chain0 MASTER (LA-AH) FFFF 24914 /clk        F   DFF1      /S_I1  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24543 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
5     chain0 MASTER (LA-AH) FFFF 24544 /clk        F   DFF1      /Z_I4  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24915 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
6     chain0 MASTER (LA-AH) FFFF 24540 /clk        F   DFF2      /N_I3  ""     (TI,QN)
             SLAVE  (FF-TE) FFFF 24545 /clk        T   DFF2      /Q_I4  ""     (TI,QN)
```

The first column in the report displays the chain cell index number, where 0 is the scan cell closest to the scan-out pin.

The second column displays the name of the scan chain the scan cell is in.

The third column displays the scan cell's sequential element type(s), primitive type, and latch triggering type during shifting. The latch triggering type can be any of the following: LE, TE, AH, AL.

The fourth column (inv) contains inversion data using F (false) to indicate no inversion difference and T (true) to indicate inversion difference. The inversion data has four parts for each scan cell memory element. The report presents these details (left-to-right) as follows:

1. Inversion of the signal between the scan chain input pin and the library cell input pin.
2. Inversion of the signal between the library cell output pin and the scan chain output pin.
3. Inversion of the signal between the library cell input pin and the scan cell memory element (primitive).
4. Inversion of the signal between the scan cell memory element (primitive) and the library cell output pin.

The following diagram illustrates the four parts of the "inv" column:

For example, scan cell 0 in the previous scan cell report for chain0 shows inversion data FFTT. This indicates that library cell DFF3 contains an inverter on the input data path as well as one on the output path as shown here:

The inversion data is useful when comparing scan cell data reported in the tool ("set_gate_report -pattern_index *n*") to values stored in the pattern file. Since different pattern formats handle inversions differently, the scan cell inversion data can also be useful in understanding differences between pattern formats.

The fifth column displays the gate index number.

The sixth column displays the name of the shift clock for the scan cell.

Note

In the rare case where a single scan cell's clock port is driven by multiple shift clocks at the same time, the tool reports the data for only one of the clocks. An example would be two different shift clocks with identical waveforms ANDed together to drive the scan cell's clock port (not recommended due to the potential for clock skew).

The seventh column shows whether the shift clock's defined inactive state corresponds to the cell's clock input value that for level-sensitive devices results in the cell becoming inactive, or for edge-triggered devices is the initial value of a capturing transition. If they correspond (no inversion), an F is listed. If they do not correspond (inversion), a T is listed. For more information on the meaning of "inactive" as it applies to scan clocks, refer to the add_clocks command description.

Note

If the scan cell is a hierarchical model, inversion is indicated with respect to the clock ports of sequential element primitives instantiated within the model, not with respect to the clock input(s) at the top level of the model.

The eighth column displays the cell name of the sequential element.

The ninth column displays the instance name of the sequential element.

The tenth column displays the library instance name. If there is no library instance name, the report shows two double-quotes in the library instance name field.

Note

To see the name of the library model corresponding to a particular scan cell, open the Browser window of DFTVisualizer with the open_visualizer -Display Browser command. In the displayed instance hierarchy, find the instance name of the scan cell. The library model is listed to the right of the instance name.

The last column displays the library cell input and output pins of the scan cell. If the scan cell input or output pin does not directly connect to the library cell boundary pin, the report shows a dash (-) in the corresponding pin field.

Example 2

The following example displays a list of all scan cells in the DFT system mode:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_system_mode analysis
report_scan_cells
```

```
-----------------------------------------------------------------------------
         Chain    Group                                                  Clock
CellNo   Name     Name     Pathname   CellName   ScanOut  Clock     Polarity
-----------------------------------------------------------------------------
0        chain1   group1   /MQ_I400   sffr       Q        clk2      (+)
1        chain1   group1   /FH_I400   sffr       QB       clk2      (+)
2        chain1   group1   /FQ_I10    sffr       QB       clk2      (+)
-        chain1   group1   /lckup1    latch      Q        clk1      (-)
3        chain1   group1   /RP_I10    sffr       Q        clk1      (+)
4        chain1   group1   /IS_I10    sffr       Q        clk1      (+)
5        chain1   group1   /CZ_I400   sffr       QB       clk1      (+)
-----------------------------------------------------------------------------
```

- The first column displays the chain cell index number, where 0-0 is the scan cell closest to the scan-out pin.
- The second column displays the chain name where the scan cell resides.
- The third column displays the group name where the scan cell resides.
- The fourth column displays the hierarchical path of the scan cell.

- The fifth column displays the library model name for the scan cell.
- The sixth column displays the scan out port of the scan cell.
- The seventh column displays the clock for the scan cell.
- The eighth column displays the polarity of the clock of the scan cell.

Example 3

The following dft -scan example adds the new column ShiftRegID/CellNo to the report when it identifies a shift register in the netlist. This column contains a tool-assigned number for the shift register ID and a cell number that indicates the order in which the flip-flops are originally connected in the shift register structures. The column contains “-/-” for those cells that are not part of a shift register.

report_scan_cells

```
-------------------------------------------------------------------------------
        Chain   Group             ShiftReg                                Clock
CellNo  Name    Name    Pathname  ID/CellNo  CellName  ScanOut Clock    Polarity
-------------------------------------------------------------------------------
0       chain1  dummy   /ud5      -/-        sff       Q         clk    (+)
1       chain1  dummy   /ud6      -/-        sff       Q         clk    (+)
2       chain1  dummy   /ud4      1/4        dff       Q         clk    (+)
3       chain1  dummy   /ud3      1/3        dff       Q         clk    (+)
4       chain1  dummy   /ud2      1/2        dff       Q         clk    (+)
5       chain1  dummy   /ud1      1/1        sff       QB        clk    (+)
-------------------------------------------------------------------------------
```

Example 4

The following dft -scan example uses the -Shift_register_flops switch to print only the shift register flops in the report:

report_scan_cells -shift_register_flops

```
-------------------------------------------------------------------------------
        Chain   Group             ShiftReg                                Clock
CellNo  Name    Name    Pathname  ID/CellNo  CellName  ScanOut  Clock  Polarity
-------------------------------------------------------------------------------
2       chain1  dummy   /ud4      1/4        dff       Q        clk    (+)
3       chain1  dummy   /ud3      1/3        dff       Q        clk    (+)
4       chain1  dummy   /ud2      1/2        dff       Q        clk    (+)
5       chain1  dummy   /ud1      1/1        sff       QB       clk    (+)
-------------------------------------------------------------------------------
```

Example 5

The following example shows use of the -Power_domain switch to report on power domains.

report_scan_cells -power_domain

```
-----------------------------------------------------------------------------------------------
CellNo  ChainName  GroupName  Pathname        CellName  ScanOut  Clock  ClockPolarity  PowerDomain
-----------------------------------------------------------------------------------------------
0        chain1     dummy   /I2/df_0001_1     sff        QB      clk       (+)          PD2
1        chain1     dummy   /I2/df_0002_1     sff        QB      clk       (+)          PD2
2        chain1     dummy   /I2/df_0003_1     sff        QB      clk       (+)          PD2
0        chain2     dummy   /I10/df_0003_1    sff        Q       clk       (+)          PD3
1        chain2     dummy   /I10/df_0001_1    sff        QB      clk       (+)          PD3
2        chain2     dummy   /I10/df_0002_1    sff        QB      clk       (+)          PD3
0        chain3     dummy   /I6               sff        QB      clk       (+)          PD1
-----------------------------------------------------------------------------------------------
```

Example 6

The following dft -scan example shows the additional output that is reported when sub-chains are encountered. Specifically, the starting and ending cell in a sub-chain are listed as a range value in the CellNo column, the clock for the first and last cell in the sub-chain are listed in the Clock column, and the polarity of each clock is listed in the Clock Polarity column.

```
-------------------------------------------------------------------------
         Chain    Group                                            Clock
CellNo   Name     Name  Pathname    CellName    ScanOut  Clock   Polarity
-------------------------------------------------------------------------
0        chain1   grp1  /usf2       SCIFTD11S10 SO       sysCLK      (+)
1        chain1   grp1  /usf1       SCIFTD11S10 SO       sysCLK      (+)
0-3      chain2   grp1  /um1        &subchain1  SO    sysCLK,sysCLK (+,+)
4-7      chain2   grp1  /uB1/um1    &subchain1  SO    sysCLK,sysCLK (+,+)
8-9      chain2   grp1  /uC1        &subchain2  so    sysCLK,sysCLK (+,+)
10       chain2   grp1  /uff2       SCIFTD11S10 SO       sysCLK      (+)
11       chain2   grp1  /uB1/uff2   SCIFTD11S10 SO       sysCLK      (+)
0-3      chain3   grp1  /um2        &subchain1  SO    MYTCLK,MYTCLK (+,+)
4-7      chain3   grp1  /uB2/um2    &subchain1  SO    MYTCLK,MYTCLK (+,+)
8-11     chain3   grp1  /uB2/um1    &subchain1  SO    MYTCLK,MYTCLK (+,+)
12-15    chain3   grp1  /uB1/um2    &subchain1  SO    MYTCLK,MYTCLK (+,+)
16-17    chain3   grp1  /uC2        &subchain2  so    MYTCLK,MYTCLK (+,+)
18       chain3   grp1  /uff1       SCIFTD11S10 SO       MYTCLK      (+)
19       chain3   grp1  /uB1/uff1   SCIFTD11S10 SO       MYTCLK      (+)
20       chain3   grp1  /uB2/uff1   SCIFTD11S10 SO       MYTCLK      (+)
21       chain3   grp1  /uB2/uff2   SCIFTD11S10 SO       MYTCLK      (+)
-------------------------------------------------------------------------
```

Related Commands

add_scan_chains

add_scan_groups

report_scan_chains

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

Mode: setup, analysis, insertion (dft -scan context only)

Usage

patterns -scan Usage

report_scan_chains [-All_blocks] [-SUbchains [-Verbose]] [-Inversion] [{> | >>} *file_pathname*]

dft -scan Usage

report_scan_chains [{> | >>} *file_pathname*]

Description

Displays a report on all the current scan chains.

The report shows the following information for each scan chain:

- Name of the scan chain
- Name of the scan chain group
- Scan chain input and output pins
- Length of the scan chain
- Name of the scan clock(s) (but only after you enable the E8 DRC in setup mode)

If the DRC E8 handling is set to anything other than Ignore when leaving Setup system mode, the command additionally lists the clocks specified to be pulsed in the shift procedure in the test procedure file. This enhanced report is only available in non-Setup modes. The clocks reported are likely a superset of the clocks that are actually used to clock the scan cells. For a precise report of the shift clocks, use the report_scan_cells command.

Arguments

- -All_blocks

Note

This switch is only for the top-level Pattern Generation Phase of modular Tessent TestKompress, where the netlist read in includes multiple EDT blocks.

An optional switch that displays information for all scan chains in every modular EDT block. Without this switch, only information defined for the current EDT block displays. If a current EDT block is not defined, the information for all EDT blocks displays. Use the current EDT block using either the add_edt_blocks or set_current_edt_block command. “EDT block” is the term used for a decompressor/compactor pair and the scan chains it

controls/observes. For more information, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

- -SUbchains

  An optional switch that displays information about the subchains of the scan chains, including the number of required serial shifts and the number of scan cells in each subchain. Only the scan chains with a subchain display.

- -Verbose

  An optional switch that displays detailed information about the scan cells in each subchain.

- -Inversion

  An optional switch that displays information about whether there is any net inversion across the entire length of the scan chain, between the scan input pin and scan output pin.
  For EDT only, this switch also displays whether there is any inversion between the decompressor output and the input of the scan chain and whether there is any inversion between the scan chain output and the compactor output. This switch can be used in all modes except Setup mode.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

The following example displays a report of all the scan chains and their scan clocks. Note that because the tool ignores the E8 DRC by default, the command does not display scan clock information.

```
ATPG> report_scan_chains

chain = chain1   group = grp1  input = /scan_in1  output = /scan_out1  length = 27
chain = chain2   group = grp1  input = /scan_in2  output = /scan_out2  length = 28
chain = chain3   group = grp1  input = /scan_in3  output = /scan_out3  length = 28
```

However, when you enable the E8 DRC in setup mode, the tool stores and reports scan clock information:

```
SETUP> set_drc_handling e8 note

SETUP> set_sytstem_mode atpg

ANALYSIS> report_scan_chains

chain = chain1   group = grp1  input = /scan_in1  output = /scan_out1  length = 27  master_clock(s) = /SCK1
chain = chain2   group = grp1  input = /scan_in2  output = /scan_out2  length = 28  master_clock(s) = /SCK2
chain = chain3   group = grp1  input = /scan_in3  output = /scan_out3  length = 28  master_clock(s) = /SCK3
```

Example 2

The following example displays a report of all the subchains.

report_scan_chains -subchains

```
chain = c1   group = g1   input = /SI0  output = /SO0  length = 11
  subchain-1 from chain "c1" cell 0 to cell 4, length = 5
  1 sub-chain in chain "c1", maximum subchain length within the chain is 5
chain = c3   group = g1   input = /SI2  output = /SO2  length = 7
  sub-chain-1 from chain "c3" cell 1 to cell 5, length = 5
  1 sub-chain in chain "c3", maximum subchain length within the chain is 5

The following 2 library models include sub-chains:
   library_name  max_subchain_length
   ------------  -------------------
   macro_sff1                      5
   reg5                            5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel
shifts.
```

Example 3

The following example indicates there is net inversion across the entire length of the scan chain, there is no inversion between the decompressor output and the input of the scan chain, and there is inversion between the scan chain output and the compactor output.

report_scan_chains -inversion

```
chain = chain1 group = grp1 input = /cpu_i/scan_in1 output =
/cpu_i/scan_out1 length = 152 inversion = Yes decompressor_inversion = No
compactor_inversion = Yes
```

Example 4

The following example shows reporting when power domains are present in the design:

report_scan_chains

```
Power domain 'PD2' chains:
---------------------------------------------------------------------------
chain = chain1  group = dummy  input = /scan_in1  output = /scan_out1
  length = 3  scan_enable = /scan_en  clock = /clk

Power domain 'PD3' chains:
---------------------------------------------------------------------------
chain = chain2   group = dummy  input = /scan_in2  output = /so2 length = 3
  scan_enable = /scan_en  clock = /clk

Power domain 'PD1' chains:
---------------------------------------------------------------------------
chain = chain3   group = dummy  input = /scan_in3  output = /scan_out2
  length = 1  scan_enable = /scan_en  clock = /clk
```

Example 5

The following example displays a detailed report of the scan cells in the subchains.

report_scan_chains -sub -verbose

```
chain = c1   group = g1  input = /SI0  output = /SO0  length = 11
  sub-chain-1 from chain "c1" cell 0 to cell 4, length = 5
----- ----- ------------- ----- ------------ ------------- --------------
cell# chain memory_type   gate# library_name instance_name (ext.pinnames)
----- ----- ------------- ----- ------------ ------------- --------------
0     c1    MASTER (FF-LE)  147 macro_sff1   /macro1  ""   (-,Q)
1     c1    MASTER (FF-LE)  148 macro_sff1   /macro1  ""   (-,-)
2     c1    MASTER (FF-LE)  149 macro_sff1   /macro1  ""   (-,-)
3     c1    MASTER (FF-LE)  150 macro_sff1   /macro1  ""   (-,-)
4     c1    MASTER (FF-LE)  151 macro_sff1   /macro1  ""   (SI,-)
1 sub-chain in chain "c1", maximum sub-chain length within the chain is 5
chain = c3   group = g1  input = /SI2  output = /SO2  length = 7
  sub-chain-1 from chain "c3" cell 1 to cell 5, length = 5
----- ----- ------------- ----- ------------ ------------- --------------
cell# chain memory_type   gate# library_name instance_name (ext.pinnames)
----- ----- ------------- ----- ------------ ------------- --------------
1     c3    MASTER (FF-LE)  164 reg5         /macro5/reg0 s_5   (-,SO)
2     c3    MASTER (FF-LE)  165 reg5         /macro5/reg0 s_4   (-,-)
3     c3    MASTER (FF-LE)  166 reg5         /macro5/reg0 s_3   (-,-)
4     c3    MASTER (FF-LE)  167 reg5         /macro5/reg0 s_2   (-,-)
5     c3    MASTER (FF-LE)  168 reg5         /macro5/reg0 x_1   (SI,-)
1 sub-chain in chain "c3", maximum sub-chain length within the chain is 5

The following 2 library models include sub-chains:
   library_name  max_subchain_length
   ------------  -------------------
   macro_sff1                      5
   reg5                            5

Total 2 sub-chains in 2 library models, which cause 4 extra parallel
shifts.
```

Related Commands

add_scan_chains
delete_scan_chains
report_scan_cells
set_parallel_load_subchains

report_scan_enable

Context: dft -scan, dft -test_points

Mode: analysis, insertion

Usage

report_scan_enable [{> | >>} *file_pathname*]

Description

Reports on scan_enable signals and associated scan chains.

The following details are reported for each scan enable signal:

- Primary input port (top-level scan enable port)
- Internal connection node (internal instance pin designated as the scan enable signal driver)
- Associated scan chain(s)

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example reports the scan_enable details for each of four scan chains.

// command: report_scan_enable

```
-------------------------------------------------------
Primary Input  Internal Connection Node  Scan Chain
-------------------------------------------------------
/scan_en     --                      c3
                                     c4
-------------------------------------------------------------
/SEN        /I_IOPADS/I_SEN/I0/X      c1
                                     c2
-------------------------------------------------------------
```

Related Commands

set_scan_enable

set_scan_enable_sharing

report_scan_groups

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

Mode: setup, analysis

Usage

report_scan_groups [{> | >>} *file_pathname*]

Description

Displays a report on all the current scan chain groups.

The report_scan_groups command provides the following information in a report for each scan chain group:

- Name of the scan chain group
- Number of scan chains within the scan chain group
- Number of shifts
- Name of the test procedure file, which contains the information for controlling the scan chains in the reported scan chain group

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of file_pathname.

Example

The following example displays a report of all the scan groups:

```
add_scan_groups group1 scanfile
add_scan_groups group2 scanfile1
report_scan_groups
```

Related Commands

add_scan_groups

delete_scan_groups

report_scan_models

Context: dft -scan

Mode: setup, analysis

Usage

report_scan_models

Description

Displays the sequential scan models currently in the scan model list.

The report_scan_models command displays sequential models which you previously added to the scan model list by using the add_scan_models command.

Examples

The following example displays all the sequential scan models from the scan model list:

```
add_scan_models d_flip_flop1 d_flip_flop2
report_scan_models
```

Related Commands

add_scan_models

delete_scan_models

report_scan_partitions

Context: dft -scan

Mode: setup, analysis

Usage

report_scan_partitions {***object_name...*** | **-Default** | **-All**} [-Expand] [{> | >>} *file_pathname*]

Description

Displays the scan partitions.

The user-specified scan partitions are reported in the same order as they are specified with the add_scan_partition command. If -expand option is specified, the pathnames of all the sequential instances placed into the scan partitions are reported. Otherwise, the scan partitions are reported in the same format they are defined.

The tool treats power domains as scan partitions during scan chain stitching. You can report the sequential cells of a power domain's scan chains using the report_scan_partitions command.

Arguments

- ***object_name***

  A required repeatable string that specifies the name(s) of the scan partition to report.

- **-Default**

  A required switch that specifies to report the default scan partition.

- **-All**

  A required switch that specifies to report all scan partitions/power domains, including user-specified scan partitions, user-specified power domains, and the default scan partition.

- -Expand

  An optional switch that specifies to report each scan partition/power domain and the pathnames of all of the sequential instances in that scan partition/power domain; newly added dedicated wrapper cells and lockup cells are indicated by the string '(new cell) following their name. If this switch is not specified, the tool reports the scan partitions in the exact format they are declared with the add_scan_partition command. If the instances are added to the partition by their container module instance, for example, the sequential instances are not reported unless this option is used.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example adds two scan partitions. The first partition, partA, is added by using exact pathnames of the sequential instances and by the container module name; the second partition, partB, is added by the container module instance of the sequential instances. The scan partitions are then reported twice, with and without the expand switch.

```
add sub chain D subch1 si so 5 mux_scan se -subclock clk
add_nonscan_instances /umodA/udff1
add_scan_partition  partA  -instance udff1 umodA/udff1 umodB/udff1 -module modD
add_scan_partition  partB  -instance umodC  -number 3

report_scan_partitions -expand

-----------------------------------------------------
ScanPartitionName       Members
-----------------------------------------------------
partA                   udff1
                        umodA/udff1
                        umodB/udff1
                        umodD/subch1 (subchain)
                        umodE/umodD/subch1 (subchain)
partB                   umodC/umodX/udff1
                        umodC/umodX/udff2
                        umodC/udff1
                        umodC/udff2
default_scan_partition  udff2
                        umodA/udff2
                        umodB/udff2
-----------------------------------------------------
```

Note that the subchain cells are not reported, rather, the entire subchain is reported with its container module instance. Also, the sequential instance /umodA/udff1 is still reported as part of the scan partition, partA, even though it is declared as a nonscan instance.

```
report_scan_partitions

-------------------------------------------------------------------------
ScanPartitionName    TotalNumCells/ScannableCells  Members
-------------------------------------------------------------------------
partA                            13/12            udff1 [instance]
                                                  umodA/udff1 [instance]
                                                  umodB/udff1 [instance]
                                                  modD [module]
partB                             4/4             umodC [instance]
default_scan_partition            3/3             <all_remaining_cells>
-------------------------------------------------------------------------
```

An extra column is printed, TotalNumCells/ScannableCells, when the *-expand* switch is not specified. The subchain cells are counted based on the length specified with the Add Sub Chain command. The sequential instances that have S1/S2 scannability rule failure or that are defined nonscan are not included in the ScannableCells section of this column.

Example 2

The first example extracts all updated information from the CPF/UPF files and reports each power domain/scan partition. The second example shows the additional information reported by including the -Expand switch.

report_scan_partitions -all

```
---------------------------------------------------------------------------
ScanPartitionName    TotalNumSequentialCells/NumScannableCells  Members
---------------------------------------------------------------------------
PD2 (power domain                  3/3                       I3 [instance]
                                                             I2 [instance]
PD3 (power domain)                 3/3                      I10 [instance]
PD1 (power domain                  1/1              <all_remaining_cells>
---------------------------------------------------------------------------
```

report scan partitions -all -expand

```
---------------------------------
ScanPartitionName   Members
---------------------------------
PD2 (power domain)  I2/df_0001_1
                    I2/df_0002_1
                    I2/df_0003_1
PD3 (power domain)  I10/df_0001_1
                    I10/df_0002_1
                    I10/df_0003_1
PD1 (power domain)  I6
---------------------------------
```

Related Commands

add_scan_partition
delete_scan_partitions
set_power_domain

report_scan_pins

Context: dft -scan

Mode: setup, analysis

Usage

report_scan_pins [{> | >>} *file_pathname*]

Description

Displays all previously assigned scan input, output, and clock names.

The report_scan_pins command displays the user-specific names that you have added to the scan input and scan output pins of the scan chains.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example displays all scan input and scan output names for the scan chains and then inserts the scan chains.

```
add_clocks 0 clk
set_system_mode analysis

add_scan_pins chain1 si so
add_scan_pins chain2 si1 so1
report_scan_pins
insert_test_logic
```

Related Commands

add_scan_pins
delete_scan_pins
echo

report_scan_volume

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_scan_volume [-External] [{> | >>} *file_pathname*]

Description

Displays the volume of scan data used by the pattern set.

The report_scan_volume command displays the volume of scan data used by the pattern set, including chain test patterns. The report format depends on the design and type of patterns, as illustrated in the examples. Because the pattern count is not a good measure of EDT compression for different scan configurations, this command gives you a metric that is equally good for EDT and regular ATPG, making it easier to compare their results.

Arguments

- -External

  An optional switch that displays the volume of scan data used by the external pattern set. Prior to using this switch, you must specify an external pattern set using the read_patterns command.

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example 1

This basic EDT design example and the examples that follow show the volume display for various types of designs and patterns.

$$\text{Scan Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

```
// ---------------------------------------------------
// Scan volume report.
// -------------------
//    channels     : 1
//    shift cycles : 371+32
// ---------------------------------------------------
// pattern    # test  # scan                   volume
// type     patterns   loads  (cell loads or unloads)
// -------  --------  ------  -----------------------
// chain test      6       6                     2418
```

Example 2

The following example is a basic EDT design where Ram_sequential or multiple-load patterns trigger an expanded format because of a difference in input and output volume:

$$\text{Input Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Scan Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

```
// ----------------------------------------------------------------------
// Scan volume report.
// -------------------
//    channels     : 1
//    shift cycles : 371+32
// ----------------------------------------------------------------------
// pattern    # test  # scan  input volume  output volume  average volume
// type     patterns   loads  (cell loads) (cell unloads)
// -------  --------  ------  ------------  -------------  --------------
// chain test      6       6          2418           2418            2418
// basic          34      34         13702          13702           13702
// clock_po        2       2           806            806             806
// ram_seq        57     223         89869          22971           56420
// clock_seq     304     304        122512         122512          122512
// multi_load     12      24          9672           4836            7254
// -------  --------  ------  ------------  -------------  --------------
// total         415     593        238979         167245          203112
```

Example 3

The following example is an EDT design where the number of input channels differs from the number of output channels:

$$\text{Input Volume} = (\#\ \text{Input Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Output Channels}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

```
// ----------------------------------------------------------------------
// Scan volume report.
// -------------------
//    input channels  : 8
//    output channels : 9
//    shift cycles    : 30
// ----------------------------------------------------------------------
// pattern    # test  # scan  input volume  output volume  average volume
// type     patterns   loads  (cell loads) (cell unloads)
// -------  --------  ------  ------------  -------------  --------------
// chain test      4       4           960           1080            1020
// basic          80      80         19200          21600           20400
// ram_seq        31     118         28320           8370           18345
// clock_seq      85      85         20400          22950           21675
// multi_load     10      30          7200           2700            4950
// -------  --------  ------  ------------  -------------  ---------------
// total         210     317         76080          56700           66390
```

Example 4

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design; no Ram_sequential or multiple-load patterns:

$$\text{Scan Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

```
// ---------------------------------------------------
// Scan volume report.
// -------------------
//    scan chains  : 10
//    shift cycles : 31
// ---------------------------------------------------
// pattern    # test  # scan                   volume
// type     patterns   loads  (cell loads or unloads)
// -------  --------  ------  -----------------------
// chain_test      1       1                      310
```

Example 5

The following example is a Tessent FastScan or Tessent TestKompress (EDT Off) design with Ram_sequential or multiple-load patterns:

$$\text{Input Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Scan Loads})$$

$$\text{Output Volume} = (\#\ \text{Scan Chains}) \times (\#\ \text{Shift Cycles}) \times (\#\ \text{Test Patterns})$$

```
// ---------------------------------------------------------------------
// Scan volume report.
// -------------------
//    scan chains  : 10
//    shift cycles : 31
// ---------------------------------------------------------------------
// pattern    # test  # scan  input volume  output volume  average volume
// type     patterns   loads  (cell loads)  (cell unloads)
// -------  --------  ------  ------------  -------------  --------------
// chain test      1       1           310            310             310
// basic          42      42         13020          13020           13020
// clock_po        1       1           310            310             310
// ram_seq         6      18          5580           1860            3720
// clock_seq      53      53         16430          16430           16430
// multi_load      3       6          1860            930            1395
// -------  --------  ------  ------------  -------------  --------------
// total         106     121         37510          32860           35185
```

Related Commands

report_edt_configurations
report_edt_pins
set_edt_options

report_seq_transparent_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_seq_transparent_procedures [-All | *procedure_name*...] [-CElls] [-Load_disturbs] [-CApture_disturbs] [{> | >>} *file_pathname*]

Description

Displays a list of seq_transparent test procedures along with the associated data that you specify.

The report_seq_transparent_procedures command displays a list of the specified seq_transparent test procedures. You can optionally display data associated with each seq_transparent test procedure by specifying the appropriate switches which are described under “Arguments”.

If you do not specify any of the data switches, the command does not provide any details about the procedures.

Arguments

- -All

  An optional switch that displays the associated data for all seq_transparent procedures. If you do not specify any other argument with the -All switch, the command simply lists the names of all the currently defined seq_transparent procedures. This is the default.

- *procedure_name*

  An optional repeatable string that specifies the names of the seq_transparent procedures whose data you want to display.

- -CElls

  An optional switch that displays the seq_transparent cells associated with the specified procedures. By default, the tools do not display these cells.

- -Load_disturbs

  An optional switch that displays the scan cells with load disturbs associated with the specified procedures. By default, the tools do not display these cells.

- -CApture_disturbs

  An optional switch that displays the scan cells, primary outputs, seq_transparent cells, and RAMs which had capture disturbs. By default, the tools do not display these items.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example displays all the seq_transparent test procedures, along with the associated scan cells that had load disturbs:

```
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 clk
set_system_mode analysis
report_seq_transparent_procedures -all -load_disturbs
```

Related Commands

report_gates

report_sequential_fault_depth

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_sequential_fault_depth [{> | >>} *file_pathname*]

Description

Displays estimates of maximum test coverage possible with different sequential depth settings.

The report_sequential_fault_depth command displays the maximum test coverage the tool estimates is possible if all relevant collapsed faults are detectable. Use this command to quickly assess the upper limits of coverage possible under optimal test conditions for various sequential depths.

The data is displayed in three columns as follows:

- Depth numbers for fifteen successive sequential depths, starting with the current depth setting. The default sequential depth upon invocation of the tool is zero, but you can change it using the -Sequential switch with the set_pattern_type command.
- Number of targeted faults for each depth
- Estimated maximum collapsed test coverage achievable under the assumption that all relevant collapsed faults can be detected. This figure is an upper bound of the coverage you can expect.

Note

Relevant faults include those the tool declared ATPG_untestable (AU) in a previous ATPG run—because the tool will retarget these faults when you increase the depth. Accordingly, the tool includes these AU faults in the row corresponding to the current depth+1.

Arguments

- *> file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example shows the output of the report_sequential_fault_depth command for a sequential depth of 4:

report_sequential_fault_depth

```
// Structural heuristic analysis of the sequential depth of faults
//
//   Set       Number         Max.
// Depth    of target     possible
//    to       faults        TC[%]
//  this      (coll.)      (coll.)
//     4         1450        89.37
//     5         7002        93.59
//     6         7852        94.42
   ...
//    15        11897        98.34
//    16        12033        98.47
//    17        12099        98.53
//    18        12159        98.59
```

Related Commands

set_pattern_type

report_sequential_instances

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_sequential_instances [-SCannable] [-NONscannable] [-DEFINED_Scan]
[-DEFINED_Nonscan] [-SUbchain] [-IDentified] [-Unidentified] [-STitched]
[-Polarity {+ | -}] [-CLock *clockname* …] [-CHain *chainname* …]
[-Library_model *modelname* …]
[-INstance *object_pathname* ... | -Module *object_name* …]
[-NOHeader] [-NOFooter] [-NOVerbose] [-Format *format_code* …] [-DRIVEN_SEN_info]
[{> | >>} *file_pathname*] [-DRIVEN_SCAN_pin_cells]

Description

Reports information and testability data for the sequential instances in the design.

You can display the sequential instances in the design using various filtering options. When you specify multiple filters, the command reports only the sequential instances that satisfy every filtering condition. If no filters are specified, the command reports all sequential instances in the design.

You can also control the output of the command, such as printing header and footer information, and formatting the output columns.

The total number of reported instances is contained in the footer of the report.

Arguments

- -SCannable

  An optional switch that reports the sequential instances that can be placed into scan chains.

- -NONscannable

  An optional switch that reports the sequential instances that cannot be placed into scan chains. The scannability condition of a sequential instance is determined by the DRC rule checker.

- -DEFINED_Scan

  An optional switch that reports the sequential instances you define as scannable.

- -DEFINED_Nonscan

  An optional switch that reports the sequential instances you define as non-scannable.

- -SUbchain

  An optional switch that reports the sequential instances contained in user-defined subchains. Note that all the cells under the parent module of a subchain are marked as subchain cells. Therefore, there may be cells marked as subchain cells even though they are not physically on the specified subchains.

- -IDentified

  An optional switch that reports the sequential instances identified as scannable by the tool. This is only valid after executing a command.

- -UNidentified

  An optional switch that reports sequential instances not identified as scannable by the tool.

- -STitched

  An optional switch that reports the sequential instances already placed in scan chains. Such instances that cannot hold the scannable or non-scannable conditions.

- -Polarity + | -

  An optional switch and a sign character that reports the sequential instances clocked by stable_low (+) or stable_high (-) clocks.

- -INstance *object_pathname* …

  An optional switch and repeatable string that reports the sequential instances that reside under specified instances.

- -Module *object_name* …

  An optional switch and repeatable string that reports the sequential instances that reside under specified modules.

- -NOHeader

  An optional switch that disables reporting of the header information in the output.

- -NOFooter

  An optional switch that disables reporting of the footer information in the output. The footer information includes the total number of reported instances.

- -NOVerbose

  An optional switch that reports the body of the report output, which is all the information except the header and the footer.

- -Format *format_code* …

  An optional switch and repeatable string that specifies which information columns to include in the report. When this switch is used, the specified columns are reported in the specified order.

  Use the format codes described in Table 7-33 to specify columns.

Table 7-33. Output Format Codes

Format_code	Column	Description
PN	Pathname	Pathname of the sequential instance.
MN	Module Name	Library module name of the sequential instance.

Table 7-33. Output Format Codes

Format_code	Column	Description
GI	Gate ID	Flattened netlist gate ID of the sequential instance. Available only in analysis mode when the flattened netlist is not freed in the process (before test logic insertion). By default, this column is not included in the output.
CN	Clock Name	Name of the primary input pin that clocks the sequential instance. If this pin is unidentified, Unknown or Test-logic displays. Test-logic is reported if you issue the set_test_logic -clock ON -reset ON … command and DRC determines that test logic is needed for the sequential element to be scannable.
CP	Clock Polarity	Clock polarity information for the sequential instance.
TY	Type	Sequential instance type: • Unknown (if in setup mode) • Scannable • Pos-scannable • Non-scannable • Ignored • Trans-latch • Constant • *chain name* or DFT (after test logic insertion) If the sequential instance is part of a scan chain, the chain name is listed. If the sequential instance is either a test logic control/observe point or a lockup cell inserted by the tool, DFT is listed.
ST	Status	Sequential instance status: • None (before scan identification process) • Unidentified (before scan identification process) • Identified (after scan identification process) • Defined-scan • Defined-nonscan • Defined-nonscan [Driven-SEN] (scan-enable pin is functionally driven) • Subchain • *scan cell number* (after test logic insertion)

- -DRIVEN_SEN_info

  An optional switch that reports sequential instances with functionally-driven scan enable pins. The scan enable pin is considered functionally driven if the driving pin is a PI pin or an output pin of a library instance that is not an inverter or buffer. The driver and driven pins can be at different levels of hierarchy where the buffers and inverters between them are transparent to the tool. When functionally-driven sequential instances are found, they are added to the non-scan instance list and a warning message displays.

  If the reported global scan enable pin (driving pin) is common to all reported sequential instances, you can define it as the global scan enable pin with the set_scan_enable command in a new tool session. Once defined, it will be used as the scan enable signal during scan insertion.

- > *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to create *or* replace the contents of a specified file with command output.

- >> *file_pathname*

  An optional redirection operator and pathname pair used at the end of the argument list to append command output to the contents of a specified file.

- -DRIVEN_SCAN_pin_cells

  An optional switch that reports sequential instances with functionally-driven scan input pins. The scan input pin is considered to be functionally driven if the driving pin is a PI pin or an output pin of a library instance that is not an inverter or buffer. The driver and driven pins can be at different levels of hierarchy where the buffers and inverters between them are transparent to the tool.

  The sequential instances with functionally-driven scan input pins are still considered as scan candidates by default, unlike the sequential instances with functionally-driven scan enable pins. The tool reports the existence of functionally-driven sequential instances to the user when it switches to analysis mode. This provides you with the option of examining instances with this switch and including them in the list of previously inserted scan chains via the add_scan_chains or add_sub_chains commands; otherwise, the tool disconnects the original connections to the scan input pins and stitches them in new scan chains along with the other scan candidates.

Example

The following example identifies sequential instances with functionally driven scan enable pins, and restores the sequential instances to the scannable cell list with the original scan enable connections preserved.

The following command sets analysis mode, runs DRC and returns a warning message when driven scan enable pins are found.

set_system_mode analysis

```
// Warning: The design includes scan cells whose scan enable pins are
// driven.
```

```
// 1) These scan cells have been added to the non-scan cell list by the
// tool
// 2) You can use the report_sequential_instances command to examine them
// 3) If a pre-routed global scan enable is used, you can define it using
// the 'set_scan_insertion' command to have these cells re-evaluated.
// 4) You can also use the delete_nonscan_instances command to preserve
// the original scan enable pin driver without specifying a global
// scan enable pin and a re-evaluation.
```

The following command reports all sequential instance types.

report_sequential_instances

```
-----------------------------------------------------------------------
          Model  Clock  Clock
PathName  Name   Name   Polarity  Type       Status
-----------------------------------------------------------------------
udff4     sff    clk    (+)        Scannable   Defined-nonscan [Driven-SEN]
uA/udff3  sff    clk    (+)        Scannable   Defined-nonscan [Driven-SEN]
uA/udff1  sff    clk    (+)        Scannable   Defined-nonscan [Driven-SEN]
uA/udff2  sff    clk    (+)        Scannable   Unidentified
-----------------------------------------------------------------------
Number of instances: 4
```

The following command reports on the sequential instances with driven scan enable pins.

report_sequential_instances -driven_sen_info

```
-------------------------------------------------------------------------------
           Model  Clock  Clock                              Sen      SenDriver
PathName   Name   Name   Polarity  Type      Status         Pinname  PinPathname
-------------------------------------------------------------------------------
udff4      sff    clk    (+)       Scannable  Defined-nonscan  SE      /sen
uA/udff3   sff    clk    (+)       Scannable  Defined-nonscan  SE      /sen
uA/udff1   sff    clk    (+)       Scannable  Defined-nonscan  SE      /uA/ux/Y
-------------------------------------------------------------------------------
Number of instances: 3
// Note: A dofile named 'delete_nonscan_instances_for_driven_sen.dofile'
//       is written out in the current directory.
```

The following command executes the dofile created by the previous example and deletes the instances from the non-scan cell list.

dofile delete_nonscan_instances_for_driven_sen.dofile

```
//  command: delete_nonscan_instances udff4
//  command: delete_nonscan_instances uA/udff3
//  command: delete_nonscan_instances uA/udff1
```

The following command reports on all sequential instances.

report_sequential_instances

```
-----------------------------------------------------------------------
              Model  Clock  Clock
PathName      Name   Name   Polarity  Type       Status
-----------------------------------------------------------------------
udff4         sff    clk    (+)       Scannable  Identified [Driven-SEN]
uA/udff3      sff    clk    (+)       Scannable  Identified [Driven-SEN]
uA/udff1      sff    clk    (+)       Scannable  Identified [Driven-SEN]
uA/udff2      sff    clk    (+)       Scannable  Identified
-----------------------------------------------------------------------
Number of instances: 4
```

The following commands insert test logic and report on the inserted sequential instances stitched into scan chains.

insert_test_logic
report_sequential_instances

```
----------------------------------------------------
              Model  Clock  Clock
PathName      Name   Name   Polarity  Type    Status
----------------------------------------------------
udff4          sff    clk    (+)       chain1  0
uA/udff3       sff    clk    (+)       chain1  1
uA/udff1       sff    clk    (+)       chain1  2
uA/udff2       sff    clk    (+)       chain1  3
----------------------------------------------------
Number of instances: 4
```

Related Commands

add_nonscan_instances
delete_nonscan_instances
echo
insert_test_logic
report_dft_check
report_scan_cells
simulate_patterns
set_scan_enable
set_system_mode

report_shift_registers

Context: dft -scan

Mode: setup, analysis

Usage

report_shift_registers [-Verbose | -Summary]

Description

Reports the identified shift registers in the design after switching to analysis mode.

The tool tries to preserve the original connections inside the identified shift registers during stitching. Therefore, this command may report the shift registers in the design differently, before and after the execution of the insert_test_logic command.

For each identified shift register, this command reports the following information:

- Length
- Hierarchical path where the shift register flip-flops reside
- First and last flip-flop instance name unless the -verbose switch is specified in which case all flip-flops in the shift registers are reported

Arguments

- -Verbose

  An optional switch that reports all of the flip-flops identified in the shift registers.

- -Summary

  An optional switch that reports a summary text without printing the flip-flops identified in the shift registers.

Example 1

The following example shows the output when neither switch is specified:

report_shift_registers

```
-------------------------------------------------------------------
               Hierarchical   SequentialCell Clock          Library
Id    Length   Path           InstanceName   Edge & Name    ModelName
-------------------------------------------------------------------
[1]   4        /              ud1            + clk          dff
                              ...
                              ud4            + clk          dff
// Number of sequential elements in design: 6
// Number of shift register flops recorded for scan insertion: 4
// => 66.67% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 1
// Longest shift register has 4 flops.
// Shortest shift register has 4 flops.
// Potential number of nonscan flops to be converted to scan cells: 1
```

```
// Potential number of scan cells to be converted to nonscan flops: 0
```

Example 2

The following example shows the output when the -verbose switch is specified:

report_shift_registers -verbose

```
-----------------------------------------------------------------------
                Hierarchical   SequentialCell Clock          Library
Id    Length    Path           InstanceName   Edge & Name    ModelName
-----------------------------------------------------------------------
[1]   4         /              ud1            + clk          dff
                               ud2            + clk          dff
                               ud3            + clk          dff
                               ud4            + clk          dff
// Number of sequential elements in design: 6
// Number of shift register flops recorded for scan insertion: 4
// => 66.67% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 1
// Longest shift register has 4 flops.
// Shortest shift register has 4 flops.
// Potential number of nonscan flops to be converted to scan cells: 1
// Potential number of scan cells to be converted to nonscan flops: 0
```

Example 3

The following example shows the output when the -summary switch is specified:

report_shift_registers -summary

```
// Number of sequential elements in design: 6
// Number of shift register flops recorded for scan insertion: 4
// => 66.67% of all sequential elements in design
// Number of shift registers recorded for scan insertion: 1
// Longest shift register has 4 flops.
// Shortest shift register has 4 flops.
// Potential number of nonscan flops to be converted to scan cells: 1
// Potential number of scan cells to be converted to nonscan flops: 0
```

Related Topics

set_shift_register_identification

report_simulation_contexts

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

report_simulation_contexts [-predefined | -user_defined]

Description

Lists the available simulation contexts and indicates the current simulation context.

The report_simulation_contexts command lists all simulation contexts. Note that the command reports available simulation contexts, some of which are only available after the tool reads a procedure file.

If you issue the command without arguments, the command reports all simulation contexts.

Arguments

- -predefined

  An optional switch that reports predefined simulation contexts, which are listed in Table 7-40 on page 1648.

- -user_defined

  An optional switch that reports user-defined simulation contexts.

Example

The following example shows how the simulation context report reflects the current set of simulation contexts:

```
ANALYSIS> set_current_simulation_context stable_load_unload
ANALYSIS> add_simulation_context myContext1
//  'myContext1' is the current simulation context.

ANALYSIS> add_simulation_context myContext2
//  'myContext2' is the current simulation context.

ANALYSIS> report_simulation_contexts
```

```
Simulation Context Report

Name                           Class
---------------------------    ------------
  myContext1                   User-defined
* myContext2                   User-defined
  stable_after_setup             Predefined
  stable_capture               Predefined
  stable_load_unload           Predefined
  stable_shift                 Predefined

* = current simulation context
```

Related Commands

add_simulation_context
copy_simulation_context
delete_simulation_contexts
get_current_simulation_context
get_simulation_context_list
set_current_simulation_context

report_simulation_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

report_simulation_forces [***obj_spec***]

Description

Lists the active forces on the specified gate_pin objects.

The report_simulation_forces command lists the simulation forces on the specified gate_pin object(s). When you issue this command without arguments, the tool lists all simulation forces of the current simulation context.

Arguments

- ***obj_spec***

  An optional value that specifies one or more gate_pin objects.

Example

The following example shows the output of the report_simulation_forces command:

ANALYSIS> **add_simulation_force [get_gate_pins I1] -value 0**
ANALYSIS> **add_simulation_force [get_gate_pins I2] -value Z**
ANALYSIS> **add_simulation_force [get_gate_pins reg12/Q] -value 1**
ANALYSIS> **report_simulation_forces**

```
Value        Object
--------     ----------
0            I1
Z            I2
1            reg12/Q
```

Related Commands

add_simulation_force
delete_simulation_forces
get_simulation_value_list
simulate_clock_pulses
simulate_forces

report_statistics

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

Mode: setup, analysis, insertion (dft -scan context only)

Usage

```
report_statistics
    [-Instance instance_pathname |
    -Model {ALL | model_name} |
    {-Hierarchy    [-LEVel integer]
                   [-INDent integer]
                   [-FORmat "format_code"]
                   [-LESsthan percentage]} ]
    [-CLOCK_domains {SUMMARY | ALL | clock_pathname...}]
    [-DEtailed_analysis [-Threshold threshold_number] ]
    [-Display] [{> | >>} file_pathname]
```

Description

Displays a detailed statistics report to the screen.

The statistics report information depends on which tool you use. The following paragraphs describe the contents of the statistics report for each tool.

The tool also reports CPU time that is the cumulative CPU time of the master process since you invoked the run.

When you use the -Instance switch, the tool's statistics report lists the following three groups of information:

- The current number of collapsed or total faults, depending on the current fault mode, in each class. By default, the fault mode is full. The report does not display fault classes with no members.
- The percentage of test coverage, fault coverage, fault coverage loss, and ATPG effectiveness for collapsed or total faults. By default, the fault mode is full, however the tool can report either the collapsed number or full number depending on the current fault mode.
- The total numbers for the following:
  - Total patterns simulated in the preceding fault simulation process. This subgroup may additionally contain total numbers for the following internal patterns sets:

    Basic scan patterns
    Clock_po patterns
    Ram_sequential patterns
    Clock_sequential patterns
  - Total patterns currently in the test pattern set

- Total CPU time

If a pattern type has no patterns, the report does not display the count for that type. If all patterns are basic patterns, it will not display any count. And it counts clock_sequential patterns that are also clock_po only as clock_sequential patterns.

When you use the -Hierarchy switch, the statistics report lists the following two groups of information for each reported hierarchical module:

- The module instance name, model name and level in the hierarchy (counting from the top level, which is 0).
- A customizable list of statistics that, by default, consists of the following:
  - Total number of faults
  - Number of unobserved (UO) faults
  - Number of Atpg untestable (AU) faults
  - Test coverage

When you issue the "set_atpg_timing On" command prior to create_patterns, the statistics report also includes delay test coverage. For more information about how the tool calculates delay test coverage, refer to the *Tessent Scan and ATPG User's Manual*.

Arguments

- -Instance *instance_pathname*

  An optional switch and string pair that reports fault statistics for a specific instance. The *instance_pathname* is the name of a circuit block whose statistics you want to report. Only fault statistics are affected by this option; pattern count statistics apply to the entire design.

- -Model ALL | *model_name*

  An optional switch and string pair that specifies reporting statistics for all models or a single named model in an ATPG library. When specifying a single model, *model_name* must match an ATPG model in the ATPG library.

- -Hierarchy

  An optional switch that allows fault statistics to be reported for each hierarchical module in the design. For each reported module, the tool displays the module instance name, model name and level in the hierarchy (counting from the top level, which is 0). By default, the tool lists the following statistics for each reported module:
  - Total number of faults
  - Number or protected faults, when applicable
  - Number of unobserved (UO) faults
  - Number of Atpg untestable (AU) faults
  - Test coverage

- Test coverage loss

-LEVel *integer*

An optional switch and integer pair that, together with the -Hierarchy switch, controls the hierarchy reporting depth. The tool reports all hierarchical levels down to and including the level you specify, counting from the top level, which is 0. If you specify "-level 1", for example, the tool will display data for the top level module and modules one level of hierarchy beneath the top level. By using this switch to restrict reporting to the upper levels of a design's hierarchy, you can reduce the command's processing time on very large designs with deep hierarchy.

-INDent *integer*

An optional switch and integer pair that, together with the -Hierarchy switch, specifies the number of additional spaces to indent the information for each subsequent level of reported hierarchy. The default indentation is four spaces. Specifying "-indent 0" removes all indentation. If the "module instance name / model name / level in the hierarchy" part of any line is greater than forty characters, the tool prints the corresponding statistics on the next line.

-FORmat "*format_code…*"

An optional switch and repeatable literal pair that, together with the -Hierarchy switch, specifies which fault information to display and the order in which to display it. Use this switch to tailor the content and arrangement of displayed data in hierarchical reports. Table 7-34 lists the format code literals this switch recognizes and the information displayed for each.

Table 7-34. Format Codes

Tessent FastScan Information (uncollapsed)	Format Code	Report Label
Total faults	#f	# faults
Fault coverage	%fc	% fault
Test coverage	%tc	% test
Test coverage loss	%tcl	% TC loss
ATPG effectiveness	%ae	% atpg
UC	uc	UC
UO	uo	UO
DS	ds	DS
DI	di	DI
PT	pt	PT
UU	uu	UU
TI	ti	TI

Table 7-34. Format Codes

Tessent FastScan Information (uncollapsed)	Format Code	Report Label
BL	bl	BL
RE	re	RE
AU	au	AU

Information displays in the same order, left to right, as the format codes in the command. The default for this switch is:

```
-format "#f uo au %tc"
```

and results in a display similar to the following:

```
                                  # faults          UO     AU      % test
-top- (my_core)        [0]          46250         2316   768      93.16%
   u10 (m3s018bo)      [1]          6240          1038    99      81.36%
   u9  (m3s015bo       [1]          4526           122    60      95.97%
        m3s016bo_3)    [2]           590             7     8      97.46%
   u12 (m3s020bo)      [1]          2146            87    33      94.40%
```

Note

You must include a space between each format code you specify, and begin and end the sequence of format codes with a double quotation mark (").

-LESsthan *percentage*

An optional switch and real number pair that, together with the -Hierarchy switch, specifies to report only module instances with test coverage less than or equal to the specified percentage. The default *percentage* is 100.0 and causes all module instances to be reported.

If the preceding example had specified “-lessthan 90.0”, the tool would have reported statistics only for module instance u10 because u10 is the only instance with test coverage less than or equal to 90%:

report_statistics -hierarchy -lessthan 90.0

```
... analyzing hierarchical statistics(Coll: 2751 faults, Full: 6240)
                              # faults          UO     AU      % test
u10 (m3s018bo)    [1]           6240        1038     99      81.36%
```

- -CLOCK_domains SUMMARY | ALL | *clock_pathname*...

  An optional switch and literal or repeatable string pair that you use for the following:

  - Reporting the summary of clock domain test coverage; or
  - Specifying a list of clocks to report the coverage(s) for the specified clock domain(s).

  Choose from the following options:

SUMMARY — A literal that reports a summary of clock domain coverage and produces a Clock Domain Summary section. For each clock domain, the tool reports the percentage of faults in the clock domain with respect to the total fault population, and reports the relevant test coverage numbers for each clock domain.

ALL — A literal that reports the detailed coverage information for all clock domains.

clock_pathnames... — A repeatable string specifying a clock domain or clock domains. The tool reports the detailed coverage information for the specified clock domains.

- -DEtailed_analysis [-Threshold *threshold_number*]

  An optional switch that reports the detailed list of the following fault sub-classes:

  - AU.CC (CELL_CONSTRAINTS)
  - AU.PC (PIN_CONSTRAINTS)
  - AU.TC (TIED_CELLS)
  - DI.EDT (EDT_LOGIC)
  - UD.AAB (ATPG_ABORT)
  - UD.EAB (EDT_ABORT)
  - UD.UNS (UNSUCCESS)

  For information about any of the previous fault sub-classes, refer to the discussion about "Fault Sub-classes" in the *Tessent Scan and ATPG User's Manual*. If the tool has not performed detailed analysis prior to this command, the tool runs the detailed analysis before reporting statistics. For an example of using the -detailed_analysis switch and then reporting on faults, refer to "Example 9" on page 1334.

  -Threshold *threshold_number*

  An optional switch and numerical pair that limits the list of untestable faults displayed. The *threshold_number* is the minimum coverage loss percentage to display. The default threshold is 0.1%. Faults having a coverage loss less than the threshold are summarized at the end of the list (see "Example 2" on page 1500).

- -Display

  An optional switch that specifies to display the following columns in the DFTVisualizer Hierarchy Browser: Test Coverage, Test Coverage Loss, Total Faults, AU, and Relevant Test Coverage. If *-Display* is specified with the *-DEtailed_analysis* switch and AU.PC and AU.TC faults exist, the AU.PC and AU.TC columns are also displayed in the Hierarchy Browser.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- *>> file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example displays a statistics report for the top level instance in a design after performing an ATPG run:

set_system_mode analysis
create_patterns
report_statistics -instance

```
                          Statistics Report
                          Transition Faults
---------------------------------------------------------------------------
Fault Classes                                #faults          #faults
                                             (total)      (total relevant)
-------------------------------------    ------------- ------------------
  FU (full)                                   1446            1424
  -----------------------------------    ------------- ------------------
  DS (det_simulation)                       24 ( 1.66%)       same ( 1.69%)
  DI (det_implication)                     326 (22.54%)       same (22.89%)
  UU (unused)                              242 (16.74%)       same (16.99%)
  AU (atpg_untestable)                     854 (59.06%)        832 (58.43%)
---------------------------------------------------------------------------
Untested Faults
  --------------------------------
  AU (atpg_untestable)
    BB  (black_boxes)                      176 (12.17%)       same (12.36%)
    PC*  (pin_constraints)                  22 ( 1.52%)       deleted
    TC*  (tied_cells)                      120 ( 8.30%)       same ( 8.43%)
    Unclassified                           536 (37.07%)       same (37.64%)
  *Use "report_statistics -detailed_analysis" for details.
---------------------------------------------------------------------------
Coverage
  --------------------------------
  test_coverage                                 29.07%             29.61%
  fault_coverage                                24.20%             24.58%
  atpg_effectiveness                           100.00%            100.00%
---------------------------------------------------------------------------
#test_patterns                                                          4
  #clock_sequential_patterns                                            4
#simulated_patterns                                                    64
CPU_time (secs)                                                       0.5
---------------------------------------------------------------------------
```

Here is how the statistics report would look if you used a “report_statistics -hierarchy -level 1” command:

```
.. analyzing hierarchical statistics ( Coll: 1066 faults, Full: 1446 )
                                          # faults    UO   AU  %  test
    -top-    (top)          [0]           1446        0  854  29.07%
    gater1   (clk_gater)    [1]             26        0   18  30.77%
    an_1     (and03)        [1]              8        0    8   0.00%
...
```

Example 2

The following example shows a statistics report with detailed analysis:

report_statistics -detailed_analysis

```
                 Statistics Report
                  Stuck-at Faults
-------------------------------------------------
Fault Classes                          #faults
                                       (total)
--------------------------------  --------------
  FU (full)                             2758
  ------------------------------  --------------
  DS (det_simulation)                 12 ( 0.44%)
  DI (det_implication)                 3 ( 0.11%)
  UU (unused)                          4 ( 0.15%)
  TI (tied)                            6 ( 0.22%)
  AU (atpg_untestable)              2733 (99.09%)
-------------------------------------------------
Untested Faults
  ------------------------------
  AU (atpg_untestable)
    TC   (tied_cells)               2730 (98.98%)
      /reg_x_7_/DF1 (647) TX         596 (21.61%)
      /reg_x_0_/DF1/DF0 (654) TX     181 ( 6.56%)
      /reg_f_7_/DF1 (656) TX         148 ( 5.37%)
      /reg_f_3_/DF1 (660) TX         148 ( 5.37%)
      /reg_f_4_/DF1 (659) TX         148 ( 5.37%)
      /reg_f_6_/DF1 (657) TX         146 ( 5.29%)
      /reg_f_2_/DF1 (661) TX         146 ( 5.29%)
      /reg_f_5_/DF1 (658) TX         146 ( 5.29%)
      /reg_f_8_/DF1 (655) TX         138 ( 5.00%)
      /reg_y_7_/DF1 (639) TX         129 ( 4.68%)
      /reg_f_0_/DF1 (663) TX         126 ( 4.57%)
      /reg_x_4_/DF1/DF0 (650) TX      75 ( 2.72%)
      /reg_x_5_/DF1/DF0 (649) TX      73 ( 2.65%)
      /reg_x_2_/DF1/DF0 (652) TX      73 ( 2.65%)
      /reg_x_3_/DF1/DF0 (651) TX      73 ( 2.65%)
      /reg_x_6_/DF1 (648) TX          72 ( 2.61%)
      /reg_f_1_/DF1 (662) TX          58 ( 2.10%)
      /reg_x_1_/DF1/DF0 (653) TX      51 ( 1.85%)
      /reg_y_6_/DF1 (640) TX          23 ( 0.83%)
      /reg_y_0_/DF1 (646) TX          23 ( 0.83%)
      /reg_y_5_/DF1 (641) TX          21 ( 0.76%)
      /reg_y_1_/DF1 (645) TX          21 ( 0.76%)
      /reg_y_2_/DF1 (644) TX          21 ( 0.76%)
      /reg_y_3_/DF1 (643) TX          21 ( 0.76%)
      /reg_y_4_/DF1 (642) TX          21 ( 0.76%)
      (Combined tied cells)           52 ( 1.89%)
    Unclassified                       3 ( 0.11%)
-------------------------------------------------
Coverage
  ------------------------------
  test_coverage                          0.55%
  fault_coverage                         0.54%
  atpg_effectiveness                   100.00%
-------------------------------------------------
```

```
#test_patterns                                  2
#simulated_patterns                            32
CPU_time (secs)                               0.3
-------------------------------------------------
```

Use the -Threshold switch to limit the detailed list of untested faults. For example:

report_statistics -detailed_analysis -threshold 5

```
                      Statistics Report
                       Stuck-at Faults
-------------------------------------------------------------
Fault Classes                                        #faults
                                                     (total)
---------------------------------------------  --------------
  FU (full)                                          2758
  ---------------------------------------------  --------------
  DS (det_simulation)                                  12 ( 0.44%)
  DI (det_implication)                                  3 ( 0.11%)
  UU (unused)                                           4 ( 0.15%)
  TI (tied)                                             6 ( 0.22%)
  AU (atpg_untestable)                               2733 (99.09%)
-------------------------------------------------------------
Untested Faults
  -----------------------------------------
  AU (atpg_untestable)
    TC   (tied_cells)                                 2730 (98.98%)
      /reg_x_7_/DF1 (647) TX                          596 (21.61%)
      /reg_x_0_/DF1/DF0 (654) TX                      181 ( 6.56%)
      /reg_f_7_/DF1 (656) TX                          148 ( 5.37%)
      /reg_f_3_/DF1 (660) TX                          148 ( 5.37%)
      /reg_f_4_/DF1 (659) TX                          148 ( 5.37%)
      /reg_f_6_/DF1 (657) TX                          146 ( 5.29%)
      /reg_f_2_/DF1 (661) TX                          146 ( 5.29%)
      /reg_f_5_/DF1 (658) TX                          146 ( 5.29%)
      /reg_f_8_/DF1 (655) TX                          138 ( 5.00%)
      (Individual tied cells below threshold)        1469 (53.26%)
      (Combined tied cells)                            52 ( 1.89%)
    Unclassified                                        3 ( 0.11%)
-------------------------------------------------------------
Coverage
  -----------------------------------------
  test_coverage                                        0.55%
  fault_coverage                                       0.54%
  atpg_effectiveness                                 100.00%
-------------------------------------------------------------
#test_patterns                                            2
#simulated_patterns                                      32
CPU_time (secs)                                         0.3
-------------------------------------------------------------
```

Example 3

The following example shows the statistics output when the DI.EDT fault classification is activated with detailed analysis specified. The DI.EDT fault classification is shown in bold font.

set_edt_finder on
report_statistics -detailed_analysis

```
                          Statistics Report
                          Stuck-at Faults
-------------------------------------------------------------------------
Fault Classes                                   #faults        #faults
                                                (total)    (total relevant)
-------------------------------------------------------------- -------------
FU (full)                                         7484          7425
 ------------------------------------------- ------------- -------------
UO (unobserved)                                  8  (0.11%      same ( 0.11%)
DS (det_simulation)                           5728  (76.54%)    same (77.14%)
DI (det_implication)                          1634  (21.83%)    same (22.01%)
PU (posdet_untestable)                           7  ( 0.09%)    same ( 0.09%)
UU (unused)                                      8  ( 0.11%)    same ( 0.11%)
TI (tied)                                        2  ( 0.03%)    same ( 0.03%)
RE (redundant)                                  35  ( 0.47%)    same ( 0.47%)
AU (atpg_untestable)                            62  ( 0.83%)       3 ( 0.04%)
-------------------------------------------------------------------------
DI Faults
 ------------------------------------------------
 DI(det_imp)
  EDT (edt_logic)                              679 ( 9.07%)    same ( 9.14%)
  Unclassified                                 955 (12.76%)    same (12.86%)
-------------------------------------------------------------------------
Untested Faults
 ------------------------------------------------------------
 AU(atpg_untestable)
  EDT (edt_blocks)                                59 ( 0.79%)  deleted
  PC (pin_constraints)                             1 ( 0.01%)  same ( 0.01%)
    (Individual pin constraints below threshold) 1 ( 0.01%)
  Unclassified                                     2 ( 0.03%)  same ( 0.03%)
 UC+UO
  AAB (atpg_abort)                                 5 ( 0.07%)  same ( 0.07%)
  UNS (unsuccess)                                  3 ( 0.04%)  same ( 0.04%)
-------------------------------------------------------------------------
Coverage
 ------------------------------------------------------
  test_coverage                                      99.01%         99.80%
  fault_coverage                                     98.42%         99.20%
  atpg_effectiveness                                 99.89%         99.89%
```

Example 4

The following example shows the statistics report for relevant coverage. A second column is added to the report to compare statistics. The excluded fault sub-classes are marked as “deleted”.

set_relevant_coverage -exclude fp
report_statistics

```
                      Statistics Report
                       Stuck-at Faults
-------------------------------------------------------------
Fault Classes                  #faults          #faults
                               (total)      (total relevant)
   ----------------------- ---------------- ----------------
   FU (full)                    8078584         8048951
   ----------------------- ---------------- ----------------
   UC (uncontrolled)          2079 ( 0.03%)    same ( 0.03%)
```

```
   UO (unobserved)           92608 ( 1.15%)    same ( 1.16%)
   DS (det_simulation)     4797074 (59.38%)    same (59.39%)
   DI (det_implication)     677274 ( 8.38%)    same ( 8.39%)
   PU (posdet_untestable)      245 ( 0.00%)    same ( 0.00%)
   PT (posdet_testable)          4 ( 0.00%)    same ( 0.00%)
   UU (unused)               21254 ( 0.26%)    same ( 0.27%)
   TI (tied)                602348 ( 7.46%)    same ( 7.47%)
   BL (blocked)             428144 ( 5.30%)    same ( 5.31%)
   RE (redundant)            12063 ( 0.15%)    same ( 0.15%)
   AU (atpg_untestable)    1445491 (17.89%) 1415858 (17.59%)
-------------------------------------------------------------
Untested Faults
   -----------------------
   AU (atpg_untestable)
   BB (black_boxes)         148646 ( 1.84%)    same ( 1.90%)
   FP (false_paths)         165611 ( 2.05%)       deleted
   Unclassified             852523 (10.55%)    same (10.59%)
UC+UO
   AAB (ATPG_abort)          88056 ( 1.09%)    same ( 1.09%)
   Unclassified               6631 ( 0.08%)    same ( 0.08%)
-------------------------------------------------------------
Coverage
   -----------------------
   test_coverage                    78.04%           79.67%
   fault_coverage                   67.77%           68.37%
   atpg_effectiveness               98.83%           98.83%
------------------------------------------------------------
#test_patterns                                        17206
   #clock_sequential_patterns                         17206
#simulated_patterns                                   18880
CPU_time (secs)                                      5579.7
-------------------------------------------------------------
```

Example 5

The following example shows the Clock Domain Summary excerpted from a statistics report:

report_statistics -clock_domains summary

```
Clock Domain Summary                    % faults      Test Coverage
                                         (total)     (total relevant)
----------------------------------------  ------------  -----------------
/pll_ref_clk                                10.06%             8.19%
/pll_ref_clk                                20.00%             0.00%
/tbcin                                       3.03%            32.76%
/reset_tbc20                                 0.01%             6.25%
/pt_clk                                      1.02%            78.73%
/refclkr                                     0.09%            37.57%
/reset_rbc                                   0.01%             6.67%
/func1                                      40.22%            25.25%
/func2                                      18.76%            13.18%
/klok1                                       0.09%            24.43%
/klok2                                       0.00%             0.00%
/klok3                                       0.00%             0.00%
/klok4                                       0.14%            27.20%
/RESET_ALL                                   0.99%             0.21%
/arm_clk2                                    0.02%            38.04%
/arm_clk3                                   27.61%            17.95%
```

```
/ddr_clk1x                                    2.57%            35.08%
/ddr_reset                                    0.27%             0.92%
/LSI_TCK                                      2.75%            19.28%
/in_hcm[12]                                   0.13%             0.33%
/in_hcm[10]                                   0.00%             0.00%
/in_hcm[7]                                    0.18%             5.07%
/in_hcm[6]                                    2.19%             1.29%
/in_hcm[5]                                    0.74%             2.06%
/in_hcm[4]                                   18.76%            13.18%
/in_hcm[3]                                    0.14%            27.20%
/in_hcm[2]                                    0.00%             0.00%
/in_hcm[1]                                    0.96%            21.75%
/in_hcm[0]                                    0.43%            21.80%
/bidi[15]                                     0.45%            21.29%
/bidi[14]                                     0.04%             2.00%
---------------------------------------------------------------------------
```

Example 6

The following example performs timing-aware ATPG using SDF timing data. Note that the statistics report includes delay test coverage:

set_system_mode analysis
read_sdf my_design.sdf
set_atpg_timing on -clock_waveform DEFAULT 3 1 1
create_patterns
report_statistics

```
                            Statisticsreport
---------------------------------------------
                            #faults     #faults
fault class                 (coll.)     (total)
-----------------------     -------     -------
FU (full)                   3370612     4741152
-----------------------     -------     -------
UC (uncontrolled)               846        1339
UO (unobserved)               41993       63027
DS (det_simulation)         2518486     3799766
DI (det_implication)         494643      516429
PU (posdet_untestable)           34          52
UU (unused)                      30          52
TI (tied)                      4270        5308
BL (blocked)                   2654        2754
AU (atpg_untestable)         307656      352425
-----------------------     -------     -------
test_coverage                 89.58%      91.19%
fault_coverage                89.39%      91.04%
atpg_effectiveness            98.73%      98.64%
-----------------------     -------     -------
delay_test_coverage           68.09%      67.39%
---------------------------------------------
#test_patterns                               3508
      #clock_sequential_patterns             3508
#simulated_patterns                         3616
CPU_time (secs)                          17372.7
---------------------------------------------
```

Example 7

The following example shows the multiple detection test coverage report at the end of the normal report_statistics output when some unlisted faults are loaded using the “read_faults -graybox” command and AU.PC faults are excluded from the relevant test coverage.

```
> read_faults top_faults.txt -retain
> read_faults core1_faults.txt -graybox -retain
> set_relevant_coverage -exclude au.pc
> report_statistics
```

The following shows the left side of the command output:

```
...
Multiple Detection Statistics
-----------------------------------------------------------------------------
Detections                                                DS Faults
   (N)                                 (Detection == N)  (total w. unlisted)
----------                             ----------------  -------------------
   1                                      0 (  0.00%)      15563 ( 17.22%)
   2                                      0 (  0.00%)      15563 ( 17.22%)
...
   8                                    135 (  0.11%)      15249 ( 16.89%)
   9                                    198 (  0.16%)      15127 ( 16.76%)
   10+                                 5238 (  4.12%)      14950 ( 16.57%)
-----------------------------------------------------------------------------
bridge_coverage_estimate
-----------------------------------------------------------------------------
```

The following shows the right side of the command output, with the Detections column included for your reference.

```
Multiple Detection Statistics
-----------------------------------------------------------------------------
Detections                 Test Coverage (Detection >= N)
(N)      (revelant w. unlisted)(total w/o unlisted)(revelant w/o unlisted)
------    --------------------  ------------------  --------------------
1              17.30%             18220 ( 12.16%)                  12.17%
2              17.30%             18220 ( 12.16%)                  12.17%
...
8              16.97%             17870 ( 11.76%)                  11.78%
9              16.84%             17735 ( 11.61%)                  11.62%
10+            16.65%             17537 ( 11.39%)                  11.40%
-----------------------------------------------------------------------------
bridge_coverage_estimate                                           15.55%
-----------------------------------------------------------------------------
```

Related Commands

set_atpg_timing

set_relevant_coverage

report_sub_chains

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_sub_chains [{> | >>} *file_pathname*]

Description

Displays a report on the scan subchains.

The report_sub_chains command transcripts the scan subchain definition(s), which includes the scan type, module or instance pathname, name of the subchain, type of subchain the subchain is stitched into (if any), defined length of the subchain, scan input of the subchain, and the scan output of the subchain.

For mux-scan, the report includes the scan enable of the subchain.

For clocked-scan, the report includes the scan clock.

For LSSD, the report includes the scan master clock and the scan slave clock.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Examples

The following example generates a report for a scan subchain:

add_sub_chains /instA subch1 /si1 /so1 10 mux_scan -sen_core sen -instance
report_sub_chains

```
mux_scan: instA subc1 10 si1 so1 sen
```

Related Commands

add_sub_chains
delete_sub_chains
echo

report_subchain_clocks

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

report_subchain_clocks ***subchain_name***

Description

Reports on subchain clocks defined with the add_subchain_clocks command.

Arguments

- ***subchain_name***

  A required string that specifies the name of a subchain.

Examples

add_sub_chains MULTIBITSFF3 chain1 SI SO 4 mux_scan S library_model
add_subchain_clocks chain1 0 RESET -reset
add_subchain_clocks chain1 0 SET -set
add_subchain_clocks 0 CK -first_cell_clock -leading_edge
add_subchain_clocks 0 CK -last_cell_clock -leading_edge
report_subchain_clocks chain1

```
// clock name type         off_state   edge
// ---------- ----         ---------   -----
// RESET     reset             0
// SET       set               0
// CK        first cell clock  0       leading edge
// CK        last cell clock   0       leading edge
```

delete_subchain_clocks chain1 CK
report_subchain_clocks chain1

```
// clock name type  off_state  edge
// ---------- ----- --------- ------
// RESET       reset 0
// SET         set   0
```

Related Commands

add_subchain_clocks

delete_subchain_clocks

report_subchain_groups

Context: dft -scan

Mode: setup, analysis

Usage

report_subchain_groups

Description

Displays a report on the subchain groups.

Transcripts the subchain group definitions used in add_subchain_group command. The definitions include the name of the subchain group, the list of subchains, and the type of the subchain group.

Examples

The following example illustrates the usage and the output format of the report_subchain_groups command.

add_subchain_group subchaingroup1 subchain1 subchain2 -fixed
add_subchain_group subchaingroup2 subchain3 subchain4 -flexible
report_subchain_groups

```
-----------------------------------------------
SubchainGroupName   Type      ListOfSubchains
-----------------------------------------------
subchaingroup1      fixed     subchain1
                              subchain2
subchaingroup2      flexible  subchain3
                              subchain4
```

Related Commands

add_subchain_group
add_sub_chains
delete_sub_chains
delete_subchain_groups
report_sub_chains

report_test_logic

Context: dft -scan

Mode: all modes

Prerequisites: Test logic or test points must be added with the insert_test_logic command.

Usage

report_test_logic [-Instance | -Module | -Summary | -Location] [{> | >>} *file_pathname*]

Description

Displays the test logic that the tool added during the scan insertion process.

The report_test_logic command displays information about the test logic that the tool added during the scan insertion process as a result of the set_test_logic command.

Arguments

- -Instance

  An optional switch that displays the list of instance pathnames and the corresponding DFT library model that the tool inserted as test logic and test points. This option also includes a summary of the number of each instance-based DFT library model that the tool inserted. This is the default.

- -Module

  An optional switch that displays the list of module names, the list of instance pathnames, and the corresponding DFT library models that the tool inserted as test logic and test points. This option also includes a summary of the number of each module-based DFT library model that the tool inserted.

- -Summary

  An optional switch that displays a summary that contains the number of each module and instance-based DFT library model that the tool inserted.

- -Location

  An optional switch that displays the pin pathname where the tool inserted test logic and identifies whether it was a result of the test logic settings or test points that you specified.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Related Commands

add_tied_signals

echo

report_test_points

Context: dft -test_points

Mode: all modes

Usage

report_test_points [-All | -Control | -Observe] [-tool_generated_only] [> | >> *file_pathname*]

Description

Displays both user-defined and system-defined test points.

You can create user-defined test points with the add_control_points and add_observe_points commands. However, the tool does not actually insert the test point circuitry until you issue the insert_test_logic command.

The report marks the system-defined test points with "[Selected]" prior to the test point information.

Arguments

- -All

  An optional switch that displays all of the information available on both control and observe test points for both the user-defined and the system-defined test points. This is the default.

- -Control

  An optional switch that displays all the test point definitions (both user- and system-defined) that are for the purpose of enabling better test coverage in design areas where, previously, the tool could not force certain state values.

- -Observe

  An optional switch that displays all the test point definitions (both user- and system-defined) that are for the purpose of enabling better test coverage by allowing the tester access to certain fault effects.

- -tool_generated_only

  An optional switch that displays only the tool-generated test points and excludes any user-defined test points.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Example

This example triggers the automatic test point identification and analysis. In addition, the location of the identified test points are reported:

set_system_mode analysis

analyze_test_points

report_test_points

Related Commands

add_tied_signals
delete_test_points
echo
insert_test_logic
report_test_logic
add_input_constraints
set_test_point_insertion_options

report_test_stimulus

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_test_stimulus
-Set {{***id#*** | ***pin_path_name***} {**0** | **1** | **Z**}}...
| **-Cycle_set** {[-FIrst_events | -LEading_edge_events | -LAst_events]
{{*id#* | *pin_path_name*} {0 | 1 | Z} *cycle#*}...}...
| **-Write** {{***id#*** | ***instance_name***} ***address_values*** [*data_values*]}...
| **-Read** {{***id#*** | ***instance_name***} ***address_values***}...
| **-RWx** {{***id#*** | ***instance_name***} ***address_values***}...
| {**-SENsitize** {{***id#*** | ***instance_name*** | ***pin_path_name***
[-Observe_at {*id#* | *instance_name* | *pin_path_name*} [-Expect {0 | 1 | X | Z}]]}}
| -Port *port_number*
[-Verbose | -Noverbose] [-PRevious] [-STore] [{> | >>} *file_pathname*]

Description

Displays the stimulus necessary to satisfy the specified set, write, or read conditions.

This command identifies how to sensitize scan chain blockage points. For example, if you first delete all scan groups and then go to the ATPG system mode, you can issue the report_test_stimulus command for possible conditions to satisfy sensitization. That is, if the blockage was at an AND gate, you can try to set an input of the gate to 1.

If test generation is successful, the stimulus necessary to satisfy the specified conditions is displayed. The stimulus for scan cells is identified by the gate index number, instance name, and cell ID number of the scan chain.

You can access the simulated values for internal gates by using the set_gate_report command with a parallel_pattern of 0.

You can also use the “report_test_stimulus -Cycle_set” command to specify the conditions at different event times or test cycles for analysis (for example, sequential controllability analysis and transition fault analysis). Each test cycle consists of at most three sets of events (clock-off, leading edge, and trailing edge) simulated in a single cycle. The following is a description of what determines the set of events within which the justified/simulated event occurs:

- When both “clock-off simulation” and “split capture” are turned off (the default unless the “set_clock_off_simulation On” and “set_split_capture_cycle On” commands are issued or set by ATPG Expert), a cycle includes one set of simulation events; that is, there are no clock-off events and the leading edge and the trailing edge are handled in one time frame. The simulation occurs at point B as shown in Figure 7-30.

Figure 7-30. set_clock_off_simulation Off and set_split_capture_cycle Off

- When “clock-off simulation” is turned on and “split capture” is turned off, a cycle includes two sets of simulation events: clock-off events, and the events handling both the leading edge and the trailing edge events. The two simulation points for this case are at points A for clock-off events and B for leading and trailing edge events with “split capture” off as shown in Figure 7-31.

Figure 7-31. set_clock_off_simulation On and set_split_capture_cycle Off

- When “clock-off simulation” is turned off and “split capture” is turned on, a cycle includes two sets of simulation events: the leading edge and trailing edge events. For this case (as shown in Figure 7-32), the simulation points are point B for the leading edge events and point C for the trailing edge events. Turning on “split capture” is necessary if a leading edge flip-flop feeds a trailing edge flip-flop in the functional path.

Figure 7-32. set_clock_off_simulation Off and set_split_capture_cycle On

- When both “clock-off simulation and” “split capture” are turned on, a cycle includes three sets of events: clock-off, the leading edge, and the trailing edge. Figure 7-33 includes the three simulation points: A for clock-off, B for leading edge, and C for trailing edge events.

Figure 7-33. set_clock_off_simulation On and Set Split Capture On

Arguments

- **-Set *id#* | *pin_path_name* 0 | 1 | Z**

  A switch with a repeatable argument and literal pair that specifies the pin and its value for which you want to generate the appropriate stimulus. You may specify multiple argument pairs with a single -Set switch.

  id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate cannot be a RAM or ROM gate.

  pin_path_name — A string that specifies the pathname of the pin you want to set.

  0 — A literal that sets the *id#* or *pin_path_name* to 0.

  1 — A literal that sets the *id#* or *pin_path_name* to 1.

  Z — A literal that sets the *id#* or *pin_path_name* to Z.

- -Cycle_set {[-FIrst_events | -LEading_edge_events | -LAst_events] {{*id#* | *pin_path_name*} {0 | 1 | Z} *cycle#*} …} …

  A switch and repeatable argument, literal, and number set that specifies the pin, its value, and the cycle to set the value at the pin for which you want to generate the appropriate stimulus.

  The events switches determine when the value is justified/simulated in the cycle. The events switches hold for all triples (pin, value, cycle) specified thereafter until another events switch is specified in the same command. For example, you can specify "report_test_stimulus -first <triple> <triple> -last <triple>" rather than "report_test_stimulus -first <triple> -first <triple> -last <triple>".

  -FIrst_events — A switch that sets the value of all pin triplets following this switch when the first events are simulated. If the "set_clock_off_simulation On" command has been issued or set by ATPG Expert, this event is the clock-off simulation; otherwise, it is always the present leading edge event simulation.

  -LEading_edge_events — A switch that sets the value of all pin triplets following this switch at the leading edge event simulation. ATPG always has one set of events, called the "leading edge," that it simulates by default in all cycles. Leading edge always refers to that one possibility.

  -LAst_events — A switch that sets the value of all pin triplets following this switch in the last set of simulation events that occur in every cycle. This is the default. If the "set_split_capture_cycle On" command was issued or set by ATPG expert, this event

is the trailing edge events simulation. By default, the value of all pins is set at the trailing edge event in the cycle if an events switch is not specified in the -Cycle_set switch. Also, when there are several -Cycle_set switches specified in the same command line, each -Cycle_set switch resets its default time frame to the trailing edge frame regardless of the time frame specified in the previous -Cycle_set switch.

id# — An integer that specifies the identification number of the gate for which you want to set the output pin. The gate id# cannot be a flattened RAM or ROM gate.

pin_path_name — A string that specifies the pathname of the pin you want to set.

0 — A literal that sets the *id#* or *pin_path_name* to 0.

1 — A literal that sets the *id#* or *pin_path_name* to 1.

Z — A literal that sets the *id#* or *pin_path_name* to Z.

cycle# — An integer that specifies the post-load capture cycle in which the event is to occur.

With named capture procedures, the first cycle after the first scan load is “1”, the second cycle after first scan load is “2”, and so on.

Without named capture procedures, “1” indicates the cycle may be anywhere in the window and can “slide” within the allocated depth. The *cycle#* earlier is always “0” if it exists, before that “-1” if it exists, and so on. The *cycle#* after “1” is always “2”, and so on. For example, if the depth is set to 1 by you or ATPG Expert, then there is only cycle “1”. If the depth is set to 2, the cycles could be numbered “0” then “1” or “1” then “2”. In this case, if you specified “0” and “1” for *cycle#*, the cycles are the first and second cycles and there is no cycle “2”. However, if you specified “1” and “2” for *cycle#*, the cycles are the first and second cycles, and there is no cycle “0”.

Note

When used without named capture procedures, -Cycle_set is not constrained from satisfying control requirements in the observation cycle(s). Therefore, it is possible for it to succeed although ATPG (stuck-fault, transition fault, MacroTest read/observe pattern, etc.) would fail, because they typically will have reserved at least one observation cycle. Such false successes can be prevented by temporarily reducing depth to artificially reserve observation cycle(s), or by scheduling a clock pulse or some other event in each observation cycle. Each solves the false success problem by creating -cycle_set compatibility with the cycle requirements imposed upon ATPG for it to succeed. If -Cycle_set fails, ATPG should also.

- **-Write *id#* | *instance_name address_values*** *data_values*

  A switch with a repeatable argument that specifies the RAM to which you want to write and, optionally, its address and data values. You may specify multiple argument triplets with a single -Write switch. The switch arguments are:

  id# — An integer that specifies the identification number of the RAM gate to which you want to write.

instance_name — A string that specifies the pathname of the RAM instance to which you want to write.

address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

data_values — An optional character string consisting of 0's, 1's, and X's that specifies the values you want to place on the RAM data lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM data lines available.

If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-Read *id#* | *instance_name address_values***

  A switch with a repeatable argument that specifies the RAM from which you want to read and, optionally, its address value. You may specify multiple argument pairs with a single -Read switch.

  id# — An integer that specifies the identification number of the RAM gate from which you want to read.

  instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

  address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

  If you do not specify the -Port switch, the command assumes the first port (port 0).

- **-RWx *id#* | *instance_name*** *address_values*

  A switch that allows a port read to be combined with a port write by combining -write and a -previous option in the commands after -rwx. Both the read and write enables of the RAM are then asserted together. The switch arguments are:

  id# — An integer that specifies the identification number of the RAM gate from which you want to read.

  instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

  address_values — A required character string consisting of 0's and 1's that specifies the values you want to place on the RAM address lines. The least significant value must be the last character in the string. The number of characters in the string must not exceed the number of RAM address lines available.

- **-SENsitize *id#* | *instance_name* | *pin_path_name***

  A switch that allows any primitive pin or pins to be targeted for sensitization; RAM pin, RAM port, ROM pin, ROM port, MUX output pin, MUX input pin, and so on. The

conditions necessary to observe the output pins (or pin) of that primitive are calculated and reported.

Note

The -SENsitize argument, like macrotest, uses a functional observation technique, unlike the basic ATPG algorithm for structural faults. Consequently, there must be a single sensitizable path from the site to be observed to some observation site (scan cell or observable PO). The sensitizable past must be through combinational gates (specifically, no sequential).

If the tool can sensitize this path, then ATPG should succeed. If the tool cannot sensitize this path, then ATPG will fail. This failure, however, indicates nothing about why a fault site is untestable, and you cannot make any conclusions about why ATPG is failing.

By issuing successive -sensitize options with different pin path names while also including the -previous option, you can find (and store with -store) the conditions that would allow a test to observe multiple sites simultaneously, even though they are driven by the outputs of different primitives. In this way, you can specify more than one pin for simultaneous observation. If a single primitive has multiple outputs, such as a RAM, and only the RAM instance name is given, all of the outputs of the RAM will be observed simultaneously, or a report of failure to sensitize will be issued. The switch arguments are:

id# — An integer that specifies the identification number of the RAM gate from which you want to read.

instance_name — A string that specifies the pathname of the RAM instance from which you want to read.

pin_path_name — A string that specifies the pathname of the pin that you want to sensitize.

-Observe_at *id# | pin_path_name | instance_name*

Specifies where the sensitized primitive output should be observed (latched for comparison). The -observe_at switch only applies to the -Sensitize switch when a single pin is specified. It is possible to specify multiple -Sensitize/-Observe_at pairings by using the -Previous option. If non-interfering paths cannot be created between all simultaneously active -Sensitize/-Observe_at points, a message is issued, and the run terminated.

-Expect 0 | 1 | X | Z

The -expect modifier also only applies if a -sensitize option is given on the same command line. It is followed by the expected output value for the primitive to be sensitized (either 0, 1, or X). The X value can also be specified using lower case x, and means unspecified output value. For example, if a RAM is being sensitized, then one output value for each output bit of the RAM must be specified if output values are given using -expect. The bits should be specified in the same order as the RAM's outputs. If a RAM named mem1 is specified with outputs Dout<7>...Dout<0>, then if the expected outputs are included, there must be 8 expected values starting with the expected value for Dout<7> as the leftmost bit. For example, 1000000x after -expect could be used to specify that the RAM's outputs are expected to be 1 for Dout<7>,

and 0 for all other bits except Dout<0>, the expected output of which is Unknown or Don't Care. If this RAM's outputs were sensitized to scan latches along an inverting path, 0111111X would be stored as the expected latched results to be scanned out as the test results.

- -Port *port_number*

  An optional switch and integer pair that specifies the identification number of the port to use for reading or writing RAM. The port identification number is zero-based; that is, the first port is "port 0". The default is 0.

- -Verbose

  An optional switch that displays the pattern that the command creates to satisfy the specified settings. This is the default.

- -Noverbose

  An optional switch that specifies to not display the pattern that the command creates to satisfy the specified settings.

- -PRevious

  An optional switch that retains the settings from the previous report_test_stimulus command and adds them to the current settings. When this switch is used, the command displays all of the retained settings. The default is to not retain the settings.

- -STore (not available in Tessent TestKompress (EDT On))

  An optional switch that places the command-created pattern in the internal test pattern area. You can then write this pattern to a file, in any format, by using the write_patterns command. The default is to not place the pattern in the internal test pattern area.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example command line is used for determining what stimulus is needed to write 11011 to address location 01011 for a RAM gate (gate ID number is 67):

set_system_mode analysis
report_test_stimulus -write 67 01011 11011

The following is an example of the display from the previous command line:

```
//  Time = 0
//  Force 1  /W1 (1)
//  Force 0  /A1[4] (2)
//  Force 1  /A1[3] (3)
```

```
//  Force 0  /A1[2] (4)
//  Force 1  /A1[1] (5)
//  Force 1  /A1[0] (6)
//  Force 0  /OE (7)
//  Force 1  /D1[0] (14)
//  Force 1  /D1[1] (15)
//  Force 0  /D1[2] (16)
//  Force 1  /D1[3] (17)
//  Force 1  /D1[4] (18)
```

Example 2

The following example command lines display the stimulus necessary to satisfy the specified conditions at different event times or test cycles. It is assumed that both “split capture” and “clock-off simulation” are turned on, and you want to determine if a slow-to-fall transition can be launched at the output pin of gate 245.

report_test_stimulus -cycle_set 245 1 0 245 0 1

In this example, logic 1 is set at the default/last event (trailing due to split capture) simulated in cycle 0, and logic 0 is set at the default/last event (trailing due to split capture) simulated in cycle 1.

report_test_stimulus -cycle_set -leading_edge 245 1 0 245 0 1

In this example, logic 1 is set at the leading edge event of cycle 0, and logic 0 is set at the leading edge event of cycle 1.

report_test_stimulus -cycle_set –first_events 245 1 0 245 0 1

In this example, logic 1 is set at the first event (clock-off) simulated in cycle 0, and logic 0 is set at the first event (clock-off in this example) of cycle 1.

report_test_stimulus -cycle_set –first_events 245 1 0 –last_events 245 0 1

In this example, logic 1 is set at the first (clock-off) event simulated in cycle 0, and logic 0 is set at the last event (trailing edge) simulated in cycle 1.

Example 3

In the following example command lines, it is assumed only split capture is turned on, and you want to determine if a slow-to-rise transition can be launched at the output pin of gate 245.

report_test_stimulus -cycle_set 245 0 1 245 1 2

In this example, logic 0 is set at the default/last (trailing due to split capture) event simulated in cycle 1, and logic 1 is set at the default/last (trailing due to split capture) event simulated in cycle 2.

report_test_stimulus -cycle_set -leading_edge 245 0 1 245 1 2

In this example, logic 0 is set at the leading edge event of cycle 1, and logic 1 is set at the leading edge event of cycle 2.

report_test_stimulus -cycle_set –first_events 245 0 1 245 1 2

In this example, logic 0 is set at the first (leading-edge) event simulated in cycle 1, and logic 1 is set at the first (leading-edge in this example) event of cycle 2.

report_test_stimulus -cycle_set –first_events 245 0 1 –last_events 245 1 2

In this example, logic 0 is set at the first (leading_edge) event simulated in cycle 1, and logic 1 is set at the last (trailing edge in this example) event simulated in cycle 2.

Related Commands

write_patterns

report_testability_data

Context: dft -edt, patterns -scan

Mode: analysis

Usage

report_testability_data **-Class *class_type*** [*filename*] [-Replace] [{> | >>} *file_pathname*]

Description

Analyzes collapsed faults for the specified fault class and displays the analysis.

The report_testability_data command identifies and displays any circuitry connections that may cause test coverage problems for the specified fault classes. In Tessent FastScan and Tessent TestKompress, the display additionally shows how many of the requested faults, if any, are protected. The display may include any of the following connection types:

- Tied or blocked by constraints
- Connected with clock lines
- Tie-x gates
- Tri-state-driver enable lines
- Non-scan latches
- Non-observable scan latches
- RAM gates
- Unresolved wired-gates
- Primary outputs that connect to clocks

If you specify a *filename* argument, the command writes to the file the list of faults with their connection information, and displays to the screen the summary of the results.

Arguments

- **-Class *class_type***

  A switch and literal pair that specifies the class of faults whose collapsed faults you want to analyze for test coverage problems. The *class_type* literal can be either a fault class code or a fault class name.

  Table 7-26 on page 1325 lists the valid fault class codes and their associated fault class names. Use either the code or the name when specifying the *class_type* literal.

- *filename*

  An optional string that specifies the name of the file to which you want to write the fault connection information. The command still displays a summary of the results to the screen.

Note

The tools write two files if there are protected faults. The *filename* file contains the data for unprotected faults and the second file (with the same name plus the suffix, ".protected") contains the information for the protected faults.

- -Replace

  An optional switch that specifies to replace the contents of the *filename* or *filename*.protected file, if they already exist.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example 1

The following example analyzes the redundant (RE) faults to identify connections causing test coverage problems and displays a summary of the results:

set_system_mode analysis
create_patterns
report_testability_data -class re

```
//  fault analysis summary of 206 faults
//  number faults connected to RAM = 196
//  number faults unclassified = 10
```

Example 2

The following example loads faults into Tessent FastScan from the external fault list file, *my_faultlist*, and applies protection to the detected faults. After an ATPG run, the example analyzes all faults, writes the fault connection information to the files, *my_testability_data* and *my_testability_data.protected*, and displays a summary of the results to the screen.

set_system_mode analysis
read_faults my_faultlist –protect
create_patterns
report_testability_data my_testability_data

```
//  fault analysis summary of 338 faults
//  number faults tied by constraints = 3
//  number faults connected to RAM = 293
//  number faults connected from RAM = 16
//  number faults unclassified = 26
//  fault analysis summary of 5592 protected faults
//  number faults connected to RAM = 2703
//  number faults connected to clock/set/reset = 500
//  number faults connected from clock = 500
//  number faults unclassified = 2389
```

Related Commands

add_faults
analyze_fault
delete_faults
read_faults
report_faults
set_fault_protection
write_faults

report_tied_signals

Context: all contexts

Mode: all modes

Usage

report_tied_signals [-Class {Full | User | System}] [{> | >>} *file_pathname*]

Description

Displays a list of the tied floating signals and pins.

The report_tied_signals command displays either the user class, system class, or full classes of tied floating signals and pins. If you do not specify a class, the command displays all the tied floating signals and pins.

Arguments

- -Class Full | User | System

  An optional switch and literal pair that specifies the source (or class) of the tied floating signals or pins which you want to display. The literal choices are as follows:

  Full — A literal that displays all the tied floating signals or pins in the user and system class. This is the default.

  User — A literal that displays only the tied floating signals or pins created using the add_tied_signals command. This includes all instance-based blackbox tied signals.

  System — A literal that displays only the netlist-described tied floating signals or pins.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Example

The following example displays the tied floating signals from the user class:

```
add_tied_signals 1 vcc vdd
report_tied_signals -class user
```

Related Commands

add_tied_signals
delete_tied_signals
report_black_boxes
set_tied_signals

report_timeplate

Context: dft -edt, dft -scan, dft -test_points, patterns -ijtag, patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup, analysis

Usage

report_timeplate {***timeplate_name*** | -**All**} [{> | >>} *file_pathname*]

Description

Displays the specified timeplate.

The report_timeplate command displays all timeplates or the specified timeplate.

Arguments

- ***timeplate_name***

  A string that specifies which timeplate to display.

- -**All**

  A switch that specifies for the tool to display all timeplates. This is the default.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of file_pathname.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of file_pathname.

Example

The following example displays the timeplate named *tp1*.

```
report_timeplate tp1
```

Related Commands

add_scan_groups
read_procfile
report_procedures
write_patterns
write_procfile

report_udfm_statistics

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

Mode: analysis

Usage

report_udfm_statistics [-INstance *pathname* ...] [-Groups *name* …]
[-GROUPS_Only {On | Off}] [-Percent {On | Off}] [-SET_Baseline]
[(> | >>) *file_name*]

Description

Reports detailed fault coverage based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU faults.

The report contains the following information as well as any additional columns you specify.

If no group is defined as a result of executing the command, by default every defect model is reported in a single row. The column headings are as follows:

- Type — Lists the UDFM type name that belongs to the defect model. For groups that collect defect models from multiple UDFM types, “---” is reported.
- Block — Identifies the object the defect model is defined on: Module, Cell, or Instance. For groups that collect different defect models, “---” is reported.
- Name — Lists the defect specific module, cell or instance name. For groups, the specified string is reported.
- Instances — Lists the number of instances of the specified module, cell, or instance identified in the design. This number is always greater zero because defect models that cannot be matched to a instance of the circuit are not reported.
- Port-Insts — Lists the number of instances where only port faults are active (that is, these instances have no cell internal faults). The information allows you to see if specific defects other than traditional port faults are targeted. The numbers in this column mean the following:
  - Port-Insts=0 — The UDFM definition contains fault definitions for cell internal defects. This means that not just port faults are targeted.
  - Port-Insts=Instances — When these two columns list the same number, the UDFM definition contains fault definitions for port faults only. This can be the case when fault sites have been added by using the add_fault_sites command.
  - Port-Insts > 0 and < Instances — Even if the UDFM definition contains a fault definition for cell internal defects, the cell-internal defects may have been dropped from the count because the flip-flop isn't part of a scan chain.
- Faults — Lists the number of faults that exist due to the specified defect model or group.

- Add-DT — Lists the difference between the stored baseline and the current status for detected faults (DT). This column displays only if there was a previous call of this function to store the baseline.
- GN — Lists the coverage gain between the stored baseline and the current status with GN=(Add-DT/(DS+DI+PT+AU+UC+UO+PU)*100). This column displays only if there was a previous call of this function to store the baseline.

Arguments

- -INstance *pathname* ...

  An optional switch and repeatable string that limits the report of UDFM statistics to one or more specific instances. The pathname argument specifies the name of a circuit block.

- -Groups *name* ...

  An optional switch and repeatable string that defines a group of UDFM defect models based on the specified *name* string(s). You can specify multiple name arguments to create multiple groups. Each *name* string creates one group. The *name* string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

  For example, the following string creates two groups: one with all defect models that contain an “AND” in their name and another with all defect models that contain an “OR” in their name.

  ```
  -Groups *AND* *OR*
  ```

  All matching models are collected regardless of whether they are a module, a cell, or an instance. Note that in the case of multiple group names, the command adds individual models to only one group, which is the first group name in the list that matches a particular model.

- -GROUPS_Only On | Off

  An optional switch and literal pair that specify how UDFM defect models that do not match any of the defined groups are reported. By default, all UDFM defect models that do not match any of the defined groups are reported in a single row. If -GROUPS_Only is set to On, this row is not created.

- -Percent On | Off

  An optional switch and literal pair that specify to report individual fault classes as an integer or as a percentage. By default, the values of the individual fault classes are reported as a percentage. When this switch is set to off, fault classes are reported as an absolute number.

- -SET_Baseline

  An optional switch that specifies for the tool to store the current coverage information as a baseline so that subsequent report_udfm_statistics commands can add information about the coverage gain. Note that the baseline becomes invalid if the fault list is modified, for example, by an add_faults or delete_faults command. The baseline also becomes invalid if you subsequently specify a different pathname with the -Instance switch.

Example 1

In this example, the command generates a detailed fault coverage report based on the UDFM defect models. The report contains columns for DS, DI, PT, TI, BL, RE, AU, UC, UO, UU, PU.

report_udfm_statistics

Note

Please note that due to space some fault class columns in the report have been removed.

```
UDFM Fault Statistic
Type   Block Name Instances Port-Insts Flts DS[%] DI[%] PT[%] TI[%] RE[%] AU[%]  UC[%] FC[%] TC[%]
------ ----- ---- --------- ---------- ---- ----- ----- ----- ----- ----- ----- ------ ----- -----
intrac Cell  OR         518          0   5000  0.00  0.10  0.00  0.00  0.00  0.20  99.70  0.10  0.10
intrac Cell  NOR        139          0   1300  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  XNOR       346          0  12000  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  NAND3      387          0   3800  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  NAND4       71          0    600  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  AND2       913          0  10000  0.00  0.00  0.00  0.00  0.00  1.10  97.60  0.00  0.00
intrac Cell  AND3        27          0    300  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  AND4        19          0    400  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  AO112        4          0    100  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
             Total     2424          0  33500  0.00  0.02  0.00  0.00  0.00  0.36  99.19  0.02  0.02
```

Example 2

In this example, the command generates a fault coverage report in which all UDFM fault models that contain the “NAN” string in their name are reported as a single row and all that contain the “AND” string in their name are reported as a single row. All others are reported one row per model.

report_udfm_statistics -groups *NAN* AND*

```
UDFM Fault Statistic
Type   Block Name  Instances Port-Insts Faults DS[%] DI[%] PT[%] TI[%] RE[%] AU[%] UC[%] FC[%] TC[
------ ----- ----  --------- ---------- ------ ---- ----- ----- ----- ----- ----- ----- ----- --
intrac Cell  OR          518          0   5000  0.00  0.10  0.00  0.00  0.00  0.20  99.70  0.10  0.10
intrac Cell  NOR         139          0   1300  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  XNOR        346          0  12000  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  AO112         4          0    100  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  *NAN*       458          0   4400  0.00  0.00  0.00  0.00  0.00  0.00 100.00  0.00  0.00
intrac Cell  AND*        957          0  10700  0.00  0.00  0.00  0.00  0.00  1.09  97.60  0.00  0.00
            Total       2424          0  33500  0.00  0.02  0.00  0.00  0.00  0.36  99.19  0.02  0.02
```

Example 3

The following example first sets a baseline for later reporting the fault coverage gain. The output of the second command displays two additional columns on the right that list the coverage gain.

report_udfm_statistics -set_baseline
...
report_udfm_statistics

```
UDFM Fault Statistics
//
Type         Block Name  Instances Port-Insts Faults  DS[%] ...  TC[%] Add-DT GN[%]
------------ ----- ----- --------- ---------- ------ ------     ------ ------ -----
intra_cell   Cell AO          1801          0 142918  99.36      99.91   5162  3.63
intra_cell   Cell MUXI4        332          0  35856 100.00     100.00   1514  4.22
intra_cell   Cell XOR3          50          0   2350  98.89     100.00     21  0.90
intra_cell   Cell AND          959          0  11660  97.64      98.96     19  0.17
intra_cell   Cell BF          1653       1653   8265  74.41      95.06     11  0.13
```

```
...
intra_cell   Cell MUXI2        204          0   4360  99.27      99.63      9  0.21
intra_cell   Cell IV         1356       1356   6780  88.79      98.02      9  0.13
intra_cell   Cell OR          515          0   7745  92.46      98.49      1  0.01
port_defects Cell DF          864         13   5220   1.02      94.61      0  0.00
                  Total     33886       3022 724779  87.71      95.69  10763  1.50
```

Related Commands

write_faults

report_version_data

Context: unspecified, all contexts

Mode: all modes

Usage

report_version_data [{> | >>} *file_pathname*]

Description

Displays the current software version information.

The report_version_data command displays information relating to the software title, version, and date.

ArgumentsV

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following is an example of the output this command displays in Tessent Shell:

report_version_data

```
Version data: Tessent Shell 2012.3   Wed Sep 05 21:31:33 GMT 2012
```

report_wrapper_cells

Context: dft -scan

Mode: analysis

Usage

report_wrapper_cells [-Summary | -Verbose]

Arguments

- -Summary

  An optional switch. For information about this switch, see the following Description.

- -Verbose

  An optional switch. For information about this switch, see the following Description.

Description

Reports information about the identified wrapper cells for each I/O port subjected to wrapper cell identification.

If no arguments are specified, the following information is reported for each I/O:

— Primary I/O port name and its direction (Input or Output).

— Maximum number of combinational logic levels between the I/O and the sequential cells identified for this I/O.

— Total number of sequential cells identified for this I/O.

— Whether this I/O was subject to the I/O Registration.

If -Verbose is specified, in addition to the above information, the following is reported:

— Identified wrapper cell path names for each I/O that succeeded the identification or “new cell” for the I/Os that failed wrapper cell identification and are subject to I/O registration.

— Both the directly-identified and feedback-identified wrapper cells. This information is only reported if the set_wrapper_chains command has either the -allow_internal_feedback or -test_points option specified.

— Identified internal feedback cells are reported as such with the “(internal feedback)” notation.

— I/Os that failed the wrapper cell identification and the reason for the failure. For a description of all possible failures, see Table 7-35.

— Total number of internal feedback gates between the directly-identified wrapper cells and internal-feedback wrapper cells. This information is only reported if the set_wrapper_chains command has either the -allow_internal_feedback or -test_points switch specified.

— Wrapper chain type indicates whether the corresponding identified wrapper cell is part of the Input or Output wrapper chain.

— Clocks controlling both the identified wrapper cells and wrapper cells to be added.

If -Summary is specified, the following information is reported:

— Total number of primary inputs.

— Total number of primary outputs.

— Total number of identified wrapper cells.

— Total number of added wrapper cells.

— Total number of design gates between the newly added dedicated wrapper cells and the logic that would have terminated the forward/backward tracing from each PI/PO during wrapper cells identification.

Note

The tracing for this reporting is limited to 1024 levels of combinational gates per PI/POs. Once the limit is reached, only the number of design gates encountered so far will be reported.

— Total number of design gates between all PI/POs for which the wrapper cells identification has succeeded, and the corresponding identified wrapper cells.

If you specify the -Verbose switch, the report includes a column that describes why an I/O failed wrapper cell identification. A description of all possible failure reasons that can be reported in the column are listed in Table 7-35 along with a description of their meaning.

Table 7-35. Reason for Failed Identification

Reason Failed Identification	Description
Max Logic Level	Exceeded the integer limit on the number of combinational logic levels between this PI/PO and the first level of sequential cells.
Max Fan In	Exceeded the integer limit on the total number of fanins at the instance.
Max Fan Out	Exceeded the integer limit on the total number of fanouts at the instance.
Combinational Feed-thru	Encountered no sequential cells or combinational logic other than buffers/inverters before reaching PO/PI: port_name.
Seq Feed-thru	Encountered no combinational logic and just one sequential cell before reaching PO/PI: port_name.
Combinational Logic Only	Encountered combinational logic only before reaching PO/PI: portname.
Black Box Instance	Encountered a blackboxed instance: instance_name.
Pre-Existing Scan Cell	Encountered a sequential cell belonging to a pre-existing scan chain: instance_name.

Table 7-35. Reason for Failed Identification

Reason Failed Identification	Description
Non-Scan Sequential Cell	Encountered a non-scan sequential cell: instance_name
Input Wrapper Cell	Encountered a sequential cell that was already identified as an output or input wrapper cell: instance_name.
Max Identified Cells	Exceeded the integer limit on the number of sequential cells identified per PO/PI.
Max Combinational Gates	Exceeded the integer limit on the number of combinational logic gates traced per PO/PI.
No Cells Identified	Unconnected or floating PO/PI.
Scan-in Pin	Port is a scan-in pin. Port is in the I/O registration list and will be registered.
Scan-out Pin	Port is a scan out pin. Port is in the I/O registration list and will be registered.
Driving a Scan-in	Port is driving a scan-in pin pin_name. Port is in the I/O registration list and will be registered.
Driven by a Scan-out	Port is driven by a scan-out pin pin_name. Port is in the I/O registration list and will be registered.
Input Subchain	Encountered a sequential cell belonging to a subchain that was assigned to an Input wrapper chain: inst-name.
Output Subchain	Encountered a sequential cell belonging to a subchain that was assigned to an Output wrapper chain: inst-name.
Subchain or Multibit Cell	Encountered a subchain or multibit cell with scan cells inside it not reachable from PIs/POs.
Input Subchain or Multibit Cell	Encountered a subchain or multibit cell that was already identified as an output wrapper cell.
Output Subchain or Multibit Cell	Encountered a subchain or multibit cell that was already identified as an input wrapper cell.
Not Traced Subchain Cell	Encountered a sequential cell belonging to a subchain whose internal scan path is not known: inst-name.
Unidentified failure	The reason for failure cannot be identified.

Example 1

The following example shows the output generated when the report_wrapper_cells command is executed with the -verbose switch and neither the -allow_internal_feedback nor -test_points switch is specified for the set_wrapper_chains command.

```
report_wrapper_cells -verbose
```

```
-----------------------------------------------------------------------------------------------------
Primary I/O  Max Logic  # Wrapper Cells  Wrapper     Wrapper  Clock     New           Reason For
Port         Level      Identified       Cells       Chain               Registration  Failed
             [32/32]    [256/256]        Identified  Type                Cell Added    Identification
-----------------------------------------------------------------------------------------------------
in3 (I)      2          2                flop1       Input    clk       No            --
                                         flop4       Input    clk
in2 (I)      1          2                flop3       Input    clk       No            --
                                         flop4       Input    clk
in1 (I)      0          1                flop2       Input    clk       No            --
in0 (I)      0          1                flop0       Input    clk       No            --
in4 (I)      0          0                new cell    Input    clk       Yes           Combinational Logic Only
in5 (I)      0          0                new cell    Input    clk       Yes           Combinational Logic Only
in6 (I)      0          0                new cell    Input    clk       Yes           Combinational Logic Only
out1 (O)     0          0                new cell    Output   test_clk  Yes           Input Wrapper Cell
out2 (O)     0          0                new cell    Output   test_clk  Yes           Combinational Logic Only
-----------------------------------------------------------------------------------------------------

The wrapper cells identification has failed for the following PIs and POs.
These PIs and POs will be I/O Registered unless excluded from the registration.
-----------------------------------------------------------------------------------------------------
in4            Input    Encountered combinational logic only before reaching PO: out2
in5            Input    Encountered combinational logic only before reaching PO: out2
in6            Input    Encountered combinational logic only before reaching PO: out2
out1           Output   Encountered a sequential cell that was already identified as an input wrapper cell: flop4
out2           Output   Encountered combinational logic only before reaching PI: in4
-----------------------------------------------------------------------------------------------------

These PIs and POs will be excluded from the I/O Registration unless they were explicitly included.
-----------------------------------------------------------------------------------------------------
out3           Output   Encountered no combinational logic and just 1 sequential cell before reaching PI: in0
-----------------------------------------------------------------------------------------------------
```

Example 2

The following example shows the output generated when the report_wrapper_cells command is executed with the -verbose switch when the -test_points option switch is specified for the set_wrapper_chains command.

report_wrapper_cells -verbose

```
-----------------------------------------------------------------------------------------------------
Primary I/O        Max Logic    #Wrapper Cells            #Internal    Wrapper      Wrapper  Clock    New
Port               Level        (Direct/Internal-Feedback)Feedback     Cells        Chain            Regist
                   [32/32]      Identified [256/256]      Gates        Identified   Type             Cell Added
-----------------------------------------------------------------------------------------------------
o[1] (O)           2            3                                      d7           Output   clk      No
                                                                       d8           Output   clk
                                                                       d9           Output   clk
o[2] (O)           0            1                                      d9           Output   clk      No
i[3] (I)           3            3/3                       4            d2           Input    clk      No
                                                                       d3           Input    clk
                                                                       d10          Input    clk
i[2] (I)           2            1/2                       2            d1           Input    clk      No
-----------------------------------------------------------------------------------------------------
```

Example 3

The following example shows the output generated when the report_wrapper_cells command is executed without the -verbose switch when the -test_points option switch is specified for the set_wrapper_chains command.

report_wrapper_cells

```
-------------------------------------------------------------------
Primary I/O  Max Logic  # Wrapper Cells  New
Port         Level      Identified       Registration
             [32/32]    [256/256]        Cell Added
-------------------------------------------------------------------
o[1] (O)     2          3                No
o[2] (O)     0          1                No
i[3] (I)     3          3                No
i[2] (I)     2          1                No
-------------------------------------------------------------------
```

Example 4

The following example shows the output generated when the report_wrapper_cells command is executed with the -verbose switch when the -allow_internal_feedback option switch is specified for the set_wrapper_chains command.

report_wrapper_cells -verbose

```
Primary I/O  Max Logic  # Wrapper Cells       Internal  Wrapper        Wrapper   Clock   New         Reason
Port         Level      (Dir/Int-Feedback)    Feedback  Cells          Chain             Registr.    Failed
             [32/32]    Identified[256/256]   Gates     Identified     Type              Cell Added  Ident.
-------------------------------------------------------------------------------------------------------------
o[1] (O)     0          1                                d2             Output    clk     No          --
i[2] (I)     2          1/2                   2          d3             Input     clk     No          --
                                                          d2(int feedb)  Output    clk
                                                          d4(int feedb)  Output    clk
i[1] (I)     1          1/1           1                  d1             Input     clk     No          --
                                                         d2(int feedb)   Output    clk
-------------------------------------------------------------------------------------------------------------
```

Related Commands

set_registered_io

set_wrapper_chains

report_write_controls

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

Mode: setup, analysis

Usage

report_write_controls [{> | >>} *file_pathname*]

Description

Displays the currently defined write control lines and their off-states.

The report_write_controls command displays the write control lines, with corresponding off-states, added using the add_write_controls command.

Arguments

- > *file_pathname*

  An optional redirection operator and pathname pair for creating *or* replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair for appending to the contents of *file_pathname*.

Example

The following example adds four write control lines and then displays a list of the control line definitions:

```
add_write_controls 0 w1 w3
add_write_controls 1 w2 w4
report_write_controls
```

Related Commands

add_write_controls
delete_write_controls
analyze_control_signals

report_xbounding

Context: dft -scan

Mode: analysis

Usage

report_xbounding {[-verbose [all | non_mcp_bounding_points | mcp_bounding_points]]}
[> | >> *file_pathname*]

Description

Reports the X-sources and optionally the scan cells used in bounding. You must issue this command to report X-bounding.

By default, this command reports a summary that includes the number of identified X-sources and also the number of control points that are required to block these X-sources from reaching any scan cells (or potential scan cells). The command can also be used to report the location of the X-bounding logic (typically muxes) as well as the control signals for the bounding logic.

Arguments

- -verbose all | non_mcp_bounding_points | mcp_bounding_points

  An optional switch and optional literal that reports the location of the scan cells used in the bounding.

  Choose one from the following:

  all — Reports all scan cells used in the bounding including non-MCP and MCP bounding points. This is the default.

  non_mcp_bounding_points — Reports only non-MCP bounding points.

  mcp_bounding_points — Reports only MCP bounding points.

- > *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for creating or replacing the contents of *file_pathname*.

- >> *file_pathname*

  An optional redirection operator and pathname pair, used at the end of the argument list, for appending to the contents of *file_pathname*.

Example

The following example shows the tool's verbose output:

report_xbounding -verbose

```
//  Added 52 control points to bound all 52 potential X sources.
MUX gate at /NEA(3) bounding mux is driven by existing flop at :
/u12/L_MSIZ_reg_7_/Q (326)
MUX gate at /NESFR(4) bounding mux is driven by existing flop at :
/u12/u1/L_SP_reg_0_/Q (323)
```

Related Commands

analyze_xbounding
set_xbounding_options
insert_test_logic
read_sdc

reset_attribute_value

Context: unspecified, all contexts

Mode: all modes

Usage

reset_attribute_value ***obj_spec*** **-name** ***attribute_name*** [-silent]

Description

Resets an attribute to its default value for the design objects specified in *obj_spec*.

Resetting an attribute differs from setting it to its default value in the fact that the reset_attribute_value also makes the attribute unspecified so it does not show up when the report_attribute and get_attribute_list commands are executed using Usage 1. The attribute's default value is specified during registration of the attribute.

This command returns the default value set for the attribute as a string.

If the attribute_name does not exist, or if an object in *obj_spec* does not exist, an error message displays unless -silent is specified.

Arguments

- ***obj_spec***

  A required value that specifies an object name, a Tcl list of object names or a collection of objects.

- **-name** ***attribute_name***

  A required switch and value pair that specify the name of the attribute.

- -silent

  An optional switch that specifies to suppress error messages if the attribute_name does not exist, or if an object in *obj_spec* does not exist.

Example

The following example shows how the command resets the added_by values for all nets to its default (“John”):

reset_attribute_value {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by
get_attribute_value_list {u1/net1 u1/net2 u2/net3 u2/net4} -name added_by

```
{John John John John}
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
report_attributes
report_attributes
set_attribute_options
set_attribute_value
unregister_attribute

reset_au_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

reset_au_faults [-REClassify]

Description

Reclassifies the faults in certain untestable categories.

Table 7-36 shows these categories.

Table 7-36. Untestable Faults that are Reclassified by reset_au_faults

Untestable Fault	Fault Reclassification
Possibly detected untestable (PU)	Possibly detected testable (PT)
ATPG untestable (AU)	Uncontrolled (UC)

Deterministic fault simulators classify some untestable faults differently depending on the algorithm. You can use the reset_au_faults command to align those (potentially) misclassified faults.

When the command changes the fault classification, the tool then has the ability to analyze and further reclassify each previously-untestable fault. Allowing the tool the ability to analyze and reclassify those particular untestable faults increases the tool's efficiency.

Arguments

- -REClassify

  An optional switch that analyzes the AU faults to identify those which can not be tested with the current settings. After resetting all AU faults, the tool will reclassify faults that are still untestable in the current mode back to AU. This switch is useful when reading in faults from a previous ATPG run, which has different settings or a different design configuration. Reclassifying the faults to AU prior to pattern generation will improve performance. If you omit this switch, the tool does not reclassify the faults and resets all AU faults. See Example 2.

Example 1

The following example sets up the tool to run the simulation with an external pattern file and resets the ATPG untestable faults so that the tool can determine their appropriate fault category:

```
read_patterns testpatterns
read_faults /user/design/fault_file -restore
reset_au_faults
simulate_patterns
```

Example 2

The following example incrementally runs ATPG in two different modes, combining the results of each session, to target additional undetected faults and increase coverage.

Dofile 1

```
// set up design to run ATPG in mode 1
add_scan_groups grp1 testproc
add_scan_chains chain1 grp1 scan_in1 x_out[0]
add_scan_chains chain2 grp1 scan_in2 scan_out1
add_scan_chains chain3 grp1 scan_in3 scan_out2
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 1 and write out faults
set_system_mode analysis
create_patterns
write_faults mode1.fault -replace
```

Dofile 2

```
// set up design to run ATPG in mode 2
add_scan_groups grp1 testproc
add_scan_chains chain4 grp1 scan_in4 y_out[2]
add_scan_chains chain5 grp1 scan_in5 x_out[5]
add_clocks 0 rst
add_clocks 0 clk
add_input_constraints /G -C1

// generate patterns from mode 2 and write out faults
set_system_mode analysis
read_faults mode1.fault  -retain
reset_au_faults -reclassify
create_patterns
read_faults mode1.fault  -merge
report_statistics
write_faults modes_1_and_2.fault -replace
```

Related Commands

read_faults

reset_di_faults

reset_bypass_chains

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

reset_bypass_chains

Description

Removes user-specified EDT bypass chain connections.

The reset_bypass_chains command removes all EDT bypass chain connections that you previously defined using the set_bypass_chains command, and reverts determination of these connections to the tool.

Example

The following example displays information on all user-defined EDT bypass chains in a design with eight scan chains and two scan channels. The example then removes the definitions, and confirms that configuration of any EDT bypass chains will now be done automatically by the tool.

report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ----------
//    1        chain1, chain5, chain6
//    2        chain2, chain3, chain4, chain7, chain8
```

reset_bypass_chains
report_bypass_chains

```
//  No user-defined bypass chains to report.
```

Related Commands

report_bypass_chains

set_bypass_chains

reset_compactor_connections

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

reset_compactor_connections

Description

Removes user-specified EDT compactor connections.

The reset_compactor_connections command removes all EDT compactor connections that you previously defined using the set_compactor_connections command, and reverts determination of these connections to the tool.

Example

The following example displays information on all user-defined compactor connections in a design with eight scan chains and two scan channels. The example then removes the user-specified definitions. Configuration of compactor connections will now be handled automatically by the tool.

report_compactor_connections

```
//    Reporting user-defined compactor connections:
//
//    Channel  Chains
//    -------  ------
//    1        chain1, chain2, chain6, chain7, chain8
//    2        chain3 - chain5
```

reset_compactor_connections
report_compactor_connections

```
//  No user-defined compactor connections to report.
```

Related Commands

report_compactor_connections

set_compactor_connections

reset_design

Context: all contexts

Mode: setup

Usage

reset_design

Description

Clears all design setup data but keeps the design and library in memory.

The command also retains the setting from the most recent set_current_design command execution.

Arguments

- None

Related Commands

delete_design

set_current_design

reset_di_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

reset_di_faults {**-ALL** | **-END_of_chains** | **-CHains** [***chain_name***...] | ***pin_pathname...***}
[-Stuck_at {01 | 0 | 1}] [-Both | -Rise | -Fall]

Description

Reclassifies DI faults to the UC category.

The reset_di_faults command reclassifies unprotected det_implication (DI) faults to the uncontrolled (UC) category. When the fault type is stuck-at or transition, this command by default resets both stuck-at-0 (or slow-to-rise for transition faults) and stuck-at-1 (or slow-to-fall for transition faults) faults at each DI fault site. You can use one of the optional switches to reset just one, rather than both, faults at each DI fault site.

Note

Unprotected faults are faults you have not used the "read_faults -Protect" or set_fault_protection command to protect. The tool never resets protected DI faults.

Use this command when you want faults that were previously classified as DI (and thus no longer targeted for detection) retargeted in the next ATPG and/or fault simulation run. For example, the tools currently declare scan path faults DI because they are tested by the chain test. This is not optimal in some cases; the end of each scan chain (after the last scan cell) is often used as a system path and it is sometimes desirable to create an at-speed transition fault test to exercise this path. The DI fault category prevents ATPG from targeting or simulating such faults. When the fault classification is changed to UC, the tool then has the ability to target the faults during ATPG and fault simulation.

- **-ALL**

  A switch that specifies for the tool to reset all unprotected DI faults to UC. This is the invocation default.

- **-END_of_chains**

  A switch that specifies to reset, for each scan chain, the unprotected scan path DI faults from the output of the last scan cell in the chain to the scan output pin.

- **-CHains** [***chain_name***...]

  A switch and optional repeating string pair that resets unprotected DI faults, including scan input and output pin faults. If you don't specify a chain_name, the tool resets faults along all scan chains. You can optionally restrict the switch operation to one or more chain_names.

- ***pin_pathname***...

  A repeating string that resets the unprotected DI faults on one more pins. The *pin_pathname* must be an instance/pin name that exists in the flattened model. These are typically model boundary pins, PIs, and POs.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies which unprotected DI faults to reset to UC. You should use this switch only if the fault type is stuck, although the tool accepts the switch regardless of fault type (see set_fault_type).

  01 — A literal specifying that for stuck-at faults the tool reset both the “stuck-at-0” and “stuck-at-1” faults; or for transition faults the tool reset both “slow-to-rise” and “slow-to-fall” faults. This is the default.

  0 — A literal specifying that for stuck-at faults the tool reset only the “stuck-at-0” faults; or for transition faults the tool reset only “slow-to-rise” faults.

  1 — A literal specifying that for stuck-at faults the tool reset only the “stuck-at-1” faults; or for transition faults the tool reset only “slow-to-fall” faults.

- -Both | -Rise | -Fall

  Optional switches that specify which unprotected transition faults to reset to UC. These switches apply to transition faults only:

  -Both - An optional switch that specifies to reset both slow-to-rise and slow-to-fall transition faults. This is the default.

  -Rise - An optional switch that specifies to reset only the slow-to-rise transition faults.

  -Fall - An optional switch that specifies to reset only the slow-to-fall transition faults.

Example 1

The following example resets all unprotected stuck-at-0 and stuck-at-1 DI faults (including clock faults) to UC.

```
reset_di_faults
```

Example 2

The following example resets all unprotected stuck-at-0 faults declared DI anywhere in the netlist.

```
reset_di_faults -stuck_at 0
```

Example 3

The following example resets stuck-at-0 and stuck-at-1 DI faults at the output of the last cell of each chain.

```
reset_di_faults -end_of_chain
```

Example 4

The following example resets only the slow-to-rise transition fault at the output of the last cell of each chain.

reset_di_faults -end_of_chain -rise

Example 5

The following example resets any DI fault anywhere along chain1.

reset_di_faults -chain chain1

Example 6

The following example resets both DI faults at pin /top/mid/bottom/Q if unprotected (any type fault).

reset_di_faults /top/mid/bottom/Q -stuck_at 01

Example 7

The following example is equivalent to the preceding example, except that it applies to transition faults only.

reset_di_faults /top/mid/bottom/Q -both

Example 8

The following example uses the defaults to reset_di_faults at the specified location (any fault type).

reset_di_faults /top/mid/bottom/Q

Example 9

The following example resets only the slow-to-rise transition fault at the specified location.

reset_di_faults /top/mid/bottom/Q -rise

Example 10

The following example resets only the stuck-at-1 fault (or slow-to-rise if transition fault) at the specified location.

reset_di_faults /top/mid/bottom/Q -stuck_at 1

Related Commands

read_faults

reset_au_faults

reset_open_pattern_set

Context: patterns -ijtag

Mode: analysis

Usage

reset_open_pattern_set

Description

Clears the content of the currently open pattern set.

After this command executes, the pattern set remains opened. The state of the ICL network is restored to the state it was in at the beginning of the pattern set which is either the end state of the previously closed pattern set when the open_pattern_set -no_initial_ireset option is used or the reset state.

Arguments

None

Return Value

None

Example

The following example clears the content of the currently open pattern set.

```
reset_open_pattern_set
```

Related Commands

close_pattern_set
get_open_pattern_set
get_pattern_set_list
open_pattern_set
report_pattern_sets
reset_open_pattern_set

reset_state

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

Mode: analysis

Usage

reset_state

Description

Resets the circuit status.

Resets all faults, except AU, TI, BL, RE and protected faults, to be undetected in order to run a new simulation; and deletes the internal patterns.

dft -scan only:

Removes all instances from both the scan identification and test point identification lists that the tool identified.

Example

The following example first performs an ATPG run, then resets the circuit status, resets the faults to be undetected, deletes the internal patterns, and then performs a new ATPG run:

```
set_system_mode analysis
create_patterns
reset_state
create_patterns
```

Related Commands

set_fault_protection

restore_design

Context: dft

Mode: insertion

Usage

restore_design

Description

Discards all edits performed in insertion mode and restores the design to its state prior to entering insertion mode.

Example

The following example creates a minor edit to the design, and then discards the edit:

```
SETUP> set_context dft -rtl
SETUP> read_verilog design1
SETUP> get_ports
{I1 I2}

SETUP> set_system_mode insertion
INSERTION> create_port I3
{I3}

INSERTION> get_ports
{I1 I2 I3}

INSERTION> restore_design
INSERTION> get_ports
{I1 I2}
```

Related Commands

write_design

return_output

Context: unspecified, all contexts

Mode: all modes

Usage

return_output ***command_line*** [-Tee]

Description

Accepts one or more tool command lines and returns the command output as a Tcl result.

Note

The return_output command has been superseded by the catch_output command. You should update your dofiles to use the new command.

Arguments

- ***command_line***

  A string that is a tool command (normally a report command) enclosed in braces { }.

- -Tee

  An optional switch that transcripts the command output as well as returning output as a Tcl result.

Example

The following example loads the output of the report_gate command into a variable named resultA.

```
set resultA [return_output {report_gate /gate3} -tee ]
// /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//      Y         O  /ff5/D  ff4/D  ff3/D /gate6b/E
puts $resultA
//  /gate3  nor02
//      A0        I  /gate2/Y
//      A1        I  /ff2/Q
//      Y         O  /ff5/D  ff4/D  ff3/D /gate6b/E
```

Related Commands

set_transcript_style

catch_output

select_display_instances

Context: all contexts

Mode: setup, analysis

Usage

select_display_instances {{{***gate_id#*** | i***nstance_name*** | ***pin_pathname***} ... [-ADd]} | **-ALl**}
[-Display {DEBug | DESign}]

Description

Selects the specified objects in the Debug and Design window(s).

By default, the command first unselects any previously selected gates. To keep previously selected gates selected, include the -Add switch.

Arguments

- ***gate_id#***

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

- ***instance_name***

  A repeatable string that specifies the name of an instance to select.

- ***pin_pathname***

  A repeatable string that specifies the name of a pin associated with a gate you want to select.

- -ADd

  An optional switch that selects specified objects without first unselecting previously-selected objects.

- **-ALl**

  A switch that selects all currently displayed gates.

- -Display {DEBug | DESign}

  A switch and literal pair that specifies the window in which this command selects objects. The following literals can be used with the -Display switch:

  DEBug
  Selects the specified objects in the Debug window. This is the default behavior.

  DESign
  Selects the specified objects in the Design window.

Examples

The following example first selects one object in the Debug window, then selects two more without unselecting the first:

select_display_instances /i$144/q
select_display_instances /i$142/q /i$141/q -add

Related Commands

open_visualizer

unselect_display_instances

set_abort_limit

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

Mode: setup, analysis

Usage

set_abort_limit ***comb_abort_limit*** [*seq_abort_limit*]

Description

Specifies the abort limit for the test pattern generator.

The set_abort_limit command performs slightly differently depending on which tool you are using. Use this command when there are some remaining undetected faults and the test coverage is still too low. By increasing the abort limit, you can allow the tool to detect those remaining undetected faults, and thereby raise the coverage. The following paragraphs describe how the command operates for each tool.

Tip: If you use ATPG Expert ("create_patterns"), you do not need to issue this command. ATPG Expert will automatically determine an appropriate abort limit.

The set_abort_limit command specifies, for combinational and/or clock sequential test generation, the maximum number of attempts the test pattern generator allows before aborting a fault. If the limit is too low, the test pattern generator may abort too many faults and fault coverage could be too low. However, if the limit is too high, it may take too much time to complete the test generation process. The default combinational abort limit value is 30.

During the test generation process for a given fault, the test pattern generator attempts combinational or ram sequential test generation first. If this fails to create a test or prove redundancy, and you specified a non-zero sequential depth by using the set_pattern_type command, the test pattern generator performs clock sequential test generation. The default abort limit for clock sequential test generation is the same as that for the combinational test generation (30).

Arguments

- ***comb_abort_limit***

  A required integer that specifies the maximum number of conflicts for each target fault that the test pattern generator allows during the combinational test generation process. The default is 30.

  If you set the combinational abort limit to 0 and the test pattern generator can perform clock sequential test generation, the generator does not perform combinational test generation.

- *seq_abort_limit*

  An optional integer that specifies the maximum number of conflicts for each target fault that the test pattern generator allows during the clock sequential test generation process. The default is the current *comb_abort_limit* default.

If you set the sequential abort limit to 0, the test pattern generator does not perform clock sequential test generation.

Example

The following example performs an ATPG run, then continues the run with a higher abort limit for the maximum number of allowed conflicts:

```
set_system_mode analysis
create_patterns
set_abort_limit 100
create_patterns
```

Related Commands

report_aborted_faults

report_faults

set_atpg_compression

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

Mode: setup, analysis

Usage

set_atpg_compression {**OFf** | {**ON** [-Limit *number*] [-Abort_limit *number*]
[-CONsecutive_fails *number*] [-SEq_merge_limit *number*]
[-Multi_clock_capture | -NOMulti_clock_capture]}}

Description

Specifies for the ATPG to perform dynamic pattern compression.

Note

Mentor Graphics recommends you use create_patterns for the smallest pattern set. The set_atpg_compression command is for specialized processes.

The set_atpg_compression command minimizes, during the ATPG run, the number of required test patterns to achieve the desired test coverage. The test pattern generator does this by attempting to detect multiple faults with a single test pattern. This is called dynamic pattern compression and typically results in a pattern set smaller than one produced by any other method except the create_patterns command.

Note

Turning dynamic ATPG pattern compression on with default settings can result in the ATPG process taking 2-3 times longer than usual. Thus, you should only use this feature if your original pattern set is unacceptably large, or when you are running the final pass to produce actual production vectors. For most efficient operation, you should use this command in conjunction with the set_decision_order -random command.

In a design with several clocks, the tool captures data using multiple clocks one after the other in separate cycles. The -Multi_clock_capture switch sets the tool to handle such designs.

Arguments

- OFf

  A required literal that disables ATPG compression during test pattern generation. This prevents the tool from using each test pattern to detect multiple faults. This is the default.

- ON

  A required literal that enables ATPG compression during test pattern generation. This allows the tool to use each test pattern to detect multiple faults.

- -Limit *number*

  An optional switch and integer pair that specifies the maximum number of faults that the test pattern generator will unsuccessfully attempt to merge with the target fault pattern. The -

Limit switch is used by the combinational compression algorithm only. Once the test pattern generator reaches the maximum number of unsuccessful mergers, the generator moves on to the next test pattern. The default is 200.

- -Abort_limit *number*

  An optional switch and integer pair that specifies the maximum number of conflicts that the test pattern generator allows for subsequent merged faults using the same pattern before aborting a fault. The default abort limit is 10.

  This switch is similar to the set_abort_limit command, however, in this instance, the set_abort_limit command limits the conflicts allowed when determining the first fault for a given pattern. This switch, on the other hand, affects the limit of conflicts allowed for subsequent merged faults for the same pattern.

- -CONsecutive_fails *number*

  An optional switch and integer pair that specifies the maximum number of *consecutive* faults that the test pattern generator will unsuccessfully attempt to merge with the target fault pattern. The -Consecutive_fails switch is used by the sequential compression algorithm only. Once the test pattern generator reaches the maximum number of *consecutive* unsuccessful mergers, the generator moves on to the next test pattern. You can increase compression effort by increasing *number*, with one exception: setting *number* to 0 specifies “no limit” on the number of consecutive unsuccessful merger attempts. This is the default.

- -SEq_merge_limit *number*

  An optional switch and integer pair that specifies the maximum number of faults that the test pattern generator will *successfully* attempt to merge with the target fault pattern. The -Seq_merge_limit switch is used by the sequential compression algorithm only. It is primarily for IDDQ and Toggle faults. By making *number* smaller, the tool more quickly terminates early compression efforts and implements fault simulation sooner. The default is 5000.

Note

Making *number* smaller may produce less compressed test sets.

- -Multi_clock_capture

  An optional switch that specifies for the tool to use a compression algorithm capable of handling multiple clock designs. Multiple clock compression uses sequential ATPG. This is the default when there are multiple clocks in a design and the sequential depth is greater than 1.

  If you include a restore_bidis statement in a clock procedure, you cannot use this switch since the procedure cannot use sequential ATPG.

Note

This switch does not support the path delay fault type.

- -NOMulti_clock_capture

  An optional switch that specifies not to use the multiple clock domain compression algorithm.

Example

The following example minimizes the number of required test patterns to achieve the desired test coverage using the set_decision_order -random switch.

```
set_decision_order -random
set_atpg_compression on
run
```

Related Commands

compress_patterns
create_patterns
delete_atpg_constraints
set_decision_order

set_atpg_fill

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

Mode: setup, analysis

Usage

set_atpg_fill { **Random** | X | 0 | 1 }

Description

Specifies how the tool should fill unspecified bits in ATPG patterns.

Every pattern generated by ATPG has deterministic bits and unspecified bits. This command allows you to specify that the tool fills those unspecified bits with random values (the default) or X's, 0's, or 1's. This command is useful for getting the test cube that the ATPG generated to test a given fault by setting the fill to X.

Arguments

- **Random** | X | 0 | 1

  A required literal that instructs the tool how to fill unspecified bits in ATPG patterns.

Example

The following example fills unspecified bits in ATPG patterns with X's:

```
set_atpg_fill X
```

set_atpg_limits

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

Mode: setup, analysis

Usage

set_atpg_limits { OFf | { [-Cpu_seconds [OFf | *integer*]] [-Pattern_count [OFf | *integer*]] [-Test_coverage [OFf | *real* | {*real integer*} ...]] } }

Description

Specifies the ATPG process limits at which the tool terminates the ATPG process.

The set_atpg_limits command determines the limitations under which the ATPG process operates. Upon invocation of the tool, all the command option limitations are off. If you set any of the limitations, and during an ATPG run the tool reaches one of those limits, the tool terminates the ATPG process. You can use any combination of the three arguments (limitation types). When you specify more than one limitation types, the tool terminates the ATPG process when any of the specified limitation types reaches its limit.

You can check the current settings of the set_atpg_limits command by using the report_environment command.

Arguments

- OFf

  An optional literal that turns off *all* ATPG process limits previously set with this command.

- -Cpu_seconds OFf | *integer*

  An optional switch and argument pair that specifies the maximum number of CPU seconds that any future ATPG process can consume before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying that there is no limit to the amount of CPU time the ATPG process consumes during an ATPG process. This is the invocation default.

  integer — A positive integer that specifies the maximum number of CPU seconds that the tool can consume during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Pattern_count [OFf | *integer*]

  An optional switch and argument pair that specifies the maximum number of test patterns that any future ATPG process can generate before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying that there is no limit to the number of test patterns the ATPG process generates during an ATPG process. This is the invocation default.

  integer — A positive integer that specifies the maximum number of test patterns that the tool can generate during an ATPG process. When the tool reaches the maximum, it terminates the ATPG process.

- -Test_coverage [OFf | *real* | {*real integer*} ...]

  An optional switch and argument pair that specifies the maximum percentage of test coverage that any future ATPG process need reach before the tool terminates the process. The argument choices are as follows:

  OFf — A literal specifying the 100 percent test coverage limit during an ATPG process. The tool terminates the ATPG process when either 100 percent coverage is attained or when the ATPG process has completed. This is the invocation default.

  real — A positive real number that specifies the maximum percentage of test coverage that the tool should achieve during an ATPG process (value range is 0 to 100.0). When the tool reaches the maximum, it terminates the ATPG process.

  {*real integer*} — One or more pairs of real and integer numbers that specifies the multiple detect coverage. For example, "-test_coverage 90.5 2" specifies the multiple detect test coverage as 90.5 for multiple detection of 2.

  When you specify multiple list pairs, such as "-test_coverage 95 1 85 5," ATPG stops at 95% coverage in the first pass targeting one detect. Then in the second pass, ATPG stops at 95% or if it completes before 95%. This continues for passes 3 and 4. Pass 5 (targeting 5-detect) stops when ATPG reaches 85% or if it completes before 85%. When you specify multiple list pairs, you cannot associate a higher detect number with a higher test coverage limit. When you specify multiple test coverage limits, the tool terminates the ATPG process only when all limits are met.

Example 1

The following example sets two of the three limits on the ATPG process and then shows the relevant setup data from the report_environment command:

set_atpg_limits -cpu_sec 500 -test_coverage 99.5 -pattern_count 100000
report_environment

```
...
atpg limits =     500.00 sec 95.50% coverage 100000 patterns
...
```

If the ATPG process reaches either of these two limits, the process terminates. Notice that the information from the report_environment command only shows the settings that are different from the invocation defaults of Off.

Example 2

The following example shows ATPG to target 5-detect with 99% test coverage as first detect limit and 95% multiple detect test coverage limit for the 2-detect to 5-detect:

```
set_multiple_detection -guaranteed_atpg_detections 5
set_atpg_limits -test_coverage 99 1 95 2
report_environment
create_patterns
```

Related Commands

report_environment

set_atpg_timing

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

Mode: setup, analysis

Usage

set_atpg_timing
[OFf | ON]
[[-CLock_waveform
{DEFAULT | *clock_pin_pathname* | *clock_gate_id*} {*clock_waveform*}]...]
[[-DELEte_clock_waveform {ALL | *clock_pin_pathname* | *clock_gate_id*}]...]
[-SLack_margin_for_fault_dropping {OFf | *slack_margin* | *slack_margin_percentage*%}]
[-TIMING_CRITICAL *timing_margin%*]

Description

Enables and defines parameters for timing-aware ATPG and fault simulation.

For complete information, see "Using Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- OFf | ON

  An optional literal that disables or enables timing-aware ATPG.

- -CLock_waveform

  An optional switch that defines a clock waveform for evaluating slack and test metrics.

  With the exception of SDQL, timing-aware ATPG uses the test clock frequency for the clock waveform you specify with this switch. The test clock frequency is the ATE testing clock frequency, which may not be the same as the normal operation frequency (System Clock).

  When you issue the report_atpg_timing command, the tool reports both the test clock and system clock.

Note

The timing information from the test procedure file is not used. You must define the clock information for all clocks in the design, even for those not used for ATPG (not used in a named capture procedure).

There are two parameters associated with the -CLock_waveform switch:

DEFAULT | *clock_pin_pathname* | *clock_gate_id*

A string that identifies the clock pin either by pin pathname or gate id. Use the DEFAULT keyword to define a waveform for all clocks that are not explicitly defined. If you define multiple waveforms for a pin or gate id, the last specified waveform is used.

clock_waveform

Defines the waveform of the clock. The clock waveform will be used for the test clock domain.

The *clock_waveform* includes three floating numbers in the following order, separated by commas:

clock_period, *offset*, *pulse_width*

The units are the same as what is specified in the SDF file, which defaults to 1 ns.

- -DELEte_clock_waveform {ALL | *clock_pin_pathname* | *clock_gate_id*}

  An optional switch and string or literal pair that deletes a clock waveform that was previously defined by the -clock_waveform switch. Use the ALL keyword to delete all explicitly defined clock waveforms. The DEFAULT clock waveform is not deleted by the ALL keyword.

- -SLack_margin_for_fault_dropping [OFF | *slack_margin* | *slack_margin_percentage*%]

  An optional switch and literal or numerical pair that specifies the fault dropping criteria in addition to the fault detection.

  The default setting for this switch is 50%. This means that for faults detected by fault simulation, if the testable minimum slack (Tms) is more than 50% of the actual test slack (Ta), then the fault is dropped. (When ATPG targets a fault, that fault is effectively dropped no matter what fault dropping criteria is in effect.)

  OFF

  Disables the additional dropping criteria, so the detection by the fault simulation is the same as the regular transition faults.

  slack_margin

  When a real number “r” is specified without a percentage sign (%), it is treated as the slack margin for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

  $$(Ta-Tms) < r$$

  Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

  slack_margin_percentage%

  If a real number “r”, ranged in [0, 100.0], is followed by a percentage sign (%), it is treated as the slack margin percentage for the fault dropping condition. A fault is dropped from the target list when the fault is detected and the following condition holds:

  $$(Ta-Tms)*100/ Ta < r$$

  Where Ta is the slack used by ATPG, and Tms is the testable minimum slack with respect to the test clock.

  If set to 100%, faults are dropped regardless of slack.

- -TIMING_CRITICAL *timing_margin%*

  An optional switch and numerical pair that selects timing-critical faults. Before running ATPG, the static longest delay path is calculated for each fault. The *timing_margin* is calculated using the following equation:

  Timing_margin% = (static_longest_delay_path) / (static_longest_delay_path + slack) * 100%

  If the *timing_margin* for a fault is greater than the specified value, the fault will be selected. For example, assume there are two faults (F1 and F2) with static longest delay paths of 8.5 ns and 9.5 ns, respectively. The clock period is 10 ns, which creates a *timing_margin* of 85% for F1 and 95% for F2. If you specify "-timing_critical 90%", F2 is selected and F1 is not.

  ATPG still runs with the complete fault list, but the *slack_margin_for_fault_dropping* is applied only to the selected timing-critical faults. Selecting a large *timing_margin* (90%) is recommended to speed up ATPG. If you want to target only selected timing-critical faults, follow the command sequence described in "Reducing run Time for Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual*.

Examples

The following example enables timing-aware ATPG and defines the clock information:

```
set_atpg_timing -clock test_clk 16000 8000 8000
set_atpg_timing -clock default 36000 18000 18000
set_atpg_timing on
```

Related Commands

report_atpg_timing

report_statistics

set_attribute_options

Context: unspecified, all contexts

Mode: all modes

Usage

set_attribute_options **-name *attribute_name_list*** [-object_types *type_list*]
[-export_during_write {auto | off | on}]
[-preserve_boundary_in_flat_model {off | on}]
[-applies_to_child_instances {off | on}]
[-display_in_gui **{**off **|** on**}]**
[-gui_marking_index *marking_index*] [-silent]

Description

Modifies options of any registered user-defined attribute.

Specifically, this command modifies the -applies_to_child_instances, -preserve_boundary_in_flat_model, -display_in_gui, and -gui_marking_index options of any registered user-defined attribute.

Arguments

- **-name *attribute_name_list***

  A required switch and value pair that specifies an attribute name or a Tcl list of attribute names whose options are to be modified.

- -object_types *type_list*

  An optional switch and value pair that specifies the object type or a Tcl list of design object types whose attribute configuration settings are to be modified. The attributes of object types not included in this list remain unaffected. If this argument is omitted, by default, the attribute options of all design object types for which the attributes are registered on are modified.

- -export_during_write auto | off | on

  An optional switch and literal pair that specifies which module and port ICL attributes are written into the ICL output when a module is written out using the write_icl command. Any attribute that does not have this value set is written out in the same form as it was read in; any attribute value set with the set_attribute_value command since being read in is ignored.

  You should use this default behavior when you want to write out the attribute as it was read in. You can modify the default behavior using the following literals:

  auto — Use *auto* when you want to update the value of an attribute that was read in with the ICL source. In this case, the value read in from ICL is written out unless an alternative value has been set using the set_attribute_value command, which will be written out instead. If the ICL source did not have an attribute definition, the value set using the set_attribute_value command is ignored and not written out when the ICL is written.

on — Use *on* when you want to update an attribute that was read in with the ICL source, or to cause a new ICL attribute definition to be created for that attribute when the ICL is written. The attribute value set using the set_attribute_value command is written out when the ICL is written out. However, the attribute definitions that were not read in from the ICL source are also written out.

off — Use *off* when you want an attribute to be ignored and omitted as the ICL is being written out. The value read in from ICL and any subsequent attribute value set using the set_attribute_value is ignored when the ICL is written out. The output ICL file will not have an attribute definition for this attribute.

- -preserve_boundary_in_flat_model off | on

  An optional switch that specifies whether to preserve certain hierarchical pins and their attributes during design flattening based on an attribute of a module, instance, port, or pin.

  For module attributes, this option preserves all pins of all instances of the module, if the attribute is set to a non-default value. For instance attributes, this option preserves all pins of the instance, if the attribute is set to a non-default value. For port attributes, this option preserves only the corresponding pin on each instance of the port's module, if the attribute is set to a non-default value. These preserved hierarchical pins become buffers in the flat model and are no-faulted. By default, hierarchical pins are not preserved.

  If you set an attribute on a pin of a design model, instance, port or pin by enabling this option, the flat model will need to be reconstructed. The attribute is not lost during flattening.

  Note that by default, only the boundaries of library cells are preserved in the flat model. (Those are the locations where faults are added for ATPG, for example.) So the boundaries of higher-level modules in the design are not preserved in the flat model by default. When you use the “-preserve_boundary_in_flat_model on” switch, each hierarchical pin and boundary of an instance is preserved in the flat model with place-holder buffers corresponding to each pin. A gate_pin object then exists in the flat model to match the hierarchical pin. The instance names of the pin and gate_pin are the same.

- -applies_to_child_instances off | on

  An optional switch that defines if the attribute is also visible on the instances below the instances or the instances of the modules on which the attribute is defined. This option can only be set to true on attributes defined for objects of type instance or module.

- -display_in_gui {off | on}

  An optional switch and literal pair that specifies whether the attribute(s) specified by the name argument is visible in the Debug and Design windows of DFTVisualizer.

  off — The attribute is not visible in the Debug and Design windows.

  on — The attribute is visible in the Debug and Design windows.

- -gui_marking_index *marking_index*

  An optional switch and value pair that specifies the marking index that DFTVisualizer uses when displaying the attribute(s). The marking index is an integer (1-10) that maps to one of

ten colors. DFTVisualizer uses the specified color as a background when it displays the attribute. The default is 1 which maps to light blue.

- -silent

  An optional switch that suppresses error messages if one or more attributes in attribute_name_list do not exist.

Example 1

The following example shows how to configure the attribute named power_domain in order for it to be visible in all child instances of the instance for which it is defined:

set_attribute_options -name power_domain -object_types instance
-applies_to_child_instances on

Example 2

This example demonstrates how an attribute value is written or not written into the ICL module representation based on the value of the set_attribute_options -export_during_write setting. This example assumes that the following ICL module has been read in:

```
Module  block {
   DataInPort din;
   DataOutPort dout;

   Attribute speed = "100";
}
```

Executing the following commands in Tessent Shell, using the default set_attribute_options settings, produces the module definition that follows:

register_attribute -name speed -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name speed -value "200"
register_attribute -name added_by -object_type icl_module -value_type string
set_attribute_value [get_icl_modules block] -name added_by -value "John"

report the ICL module with the default attribute export_during_write setting
report_icl_module block

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "100";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options *auto* setting, produces the module definition that follows:

set the attribute to auto export
set_attribute_options -name speed -export_during_write auto
set_attribute_options -name added_by -export_during_write auto
report_icl_module block

```
Module  block {
   DataInPort din;
```

```
   DataOutPort dout;
   Attribute speed = "200";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options *on* setting, produces the module definition that follows:

```
# set the attribute to always export
set_attribute_options -name speed -export_during_write on
set_attribute_options -name added_by -export_during_write on
report_icl_module block
```

```
Module  block {
   DataInPort din;
   DataOutPort dout;
   Attribute speed = "200";
   Attribute added_by = "John";
}
```

Executing the following additional commands in Tessent Shell, using the set_attribute_options *off* setting, produces the module definition that follows:

```
# set the attribute to always export
set_attribute_options -name speed -export_during_write off
set_attribute_options -name added_by -export_during_write off
report_icl_module block
```

```
Module  block {
   DataInPort din;
   DataOutPort dout;
}
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
register_attribute
report_attributes
reset_attribute_value
set_attribute_value
unregister_attribute

set_attribute_value

Context: unspecified, all contexts

Mode: all modes

Usage

set_attribute_value ***object_spec*** **-name** ***attribute_name*** **-value** ***attribute_value*** [-silent]

Description

Sets an attribute's value for the objects specified in object_spec.

This command returns a collection of objects on which the attribute exists.

Arguments

- ***object_spec***

  A required value that specifies an object name, a Tcl list of object names, or a collection of objects.

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute to set.

- **-value** ***attribute_value***

  A required switch and value pair that specifies the value to which to set the attribute.

- -silent

  An optional switch that suppresses error messages.

Example 1

The following example sets the is_hard_module attribute to true for all modules whose names starts with mem_.

set_attribute_value [get_modules mem_*] -name is_hard_module

```
{mem_a mem_b mem_c}
```

Example 2

This example sets the user-defined attribute clock_domain to clka on pin u1/u2/ck.

set_attribute_value u1/u2/ck -name clock_domain -value clka

```
{u1/u2/ck}
```

Example 3

The following example excludes clk1 and clk2 from X-constraining during core-level retargetable pattern generation of the CPU core.

```
 patterns -scan
set_current_mode -type internal
read_verilog CPU.v
read_cell_library atpglib
set_current_design
set_attribute_value {clk1 clk2} -name is_excluded_from_isolation_constraints
set_system_mode analysis
write_core_description CPU.tcd -replace
create_patterns
write_patterns CPU_stuck.retpat.qz -replace
write_faults CPU_stuck.faults.qz -replace
...
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
register_attribute
reset_attribute_value
set_attribute_options
report_attributes

set_au_analysis

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

Mode: setup, analysis

Usage

set_au_analysis {**ON** | **OFf**}

Description

Specifies whether the ATPG uses the ATPG untestable information to place ATPG untestable faults directly in the AU fault class.

The set_au_analysis command specifies whether the ATPG process can use the ATPG untestable information. Upon invocation of the tool, the AU analysis is set to On; therefore, the ATPG process uses the ATPG untestable information. Once the tool places a fault in the AU fault class, it removes the fault from the active fault list and does not simulate it. This prevents the ATPG process from identifying the faults as possibly-detected during an ATPG run. However, you may use a switch on the compress_patterns command to identify possible detections of AU faults.

Arguments

- **ON**

  A literal that specifies for the tool to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes. This is the invocation default.

- **OFf**

  A literal that specifies for the tool not to use the ATPG untestable information to place ATPG untestable faults directly in the AU fault class during any future ATPG processes.

Example

The following example specifies not to use ATPG untestable information during the ATPG run:

```
set_system_mode analysis
set_au_analysis off
create_patterns
```

Related Commands

compress_patterns
delete_atpg_constraints
read_faults

set_bidi_gating

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_bidi_gating [**OFf | ON | Scan**] [-Control {SEn | TEn}] [-Direction Input | Output] [-Top {ALl | *primary_bidi_pin*...}] [-Force_gating]

Description

Specifies how bidirectional (bidi) pins are controlled during scan chain shifting to prevent potential bus contention or to ensure an on/off state during testing.

When enabled, test logic is inserted for bidi pins as necessary to control the enable signal and/or the input/output direction as specified.

By default, when the enable signal of a bidi pin is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

You can also specify which enable signal (TEN or SEN) enables bidi pins.

Arguments

- **OFf | ON | Scan**

  Required literal that specifies whether to insert test logic to control the enable lines of bidi pins during scan chain shifting. For more information on scan chain shifting, see "Enabling Test Logic Insertion" in the *Tessent Scan and ATPG User's Manual*. Literal options include:

  OFf — no test logic is inserted to control bidi pins. Default setting.

  ON — test logic is inserted as necessary to control bidi pins.

  Scan — inserts test logic on the scan I/O bidi pins to control the direction of the bidi pin for scan shifting and ensure the success of scan chain tracing. Scan output bidi pins are gated to be in output mode while all other bidi pins are gated to be in input mode (Z state on the tester).

- -Control SEn | TEn

  An optional switch and literal pair that specifies the enable signal used to control bidi pins. Options include:

  SEn — scan_enable signal. Default setting.

  TEn — test_enable signal.

- -Direction Input | Output

  An optional switch and literal pair that specifies the direction of the bidi pins specified by the -Top switch. Options include:

Input — bidi pins are gated so they are in input mode. Default setting.

Output — bidi pins are gated so they are in output mode.

- -Top ALl | *primary_bidi_pin*...

  An optional switch and literal or repeatable string pair that specifies which bidi pins are controlled with the specified enable signal. Options include:

  ALl — all bidi pins. Default setting.

  primary_bidi_pin — a specified primary bidi pin. Test logic is inserted to ensure that these bidi pins are controlled as specified.

- -Force_gating

  An optional switch that adds test logic to the enable lines of bidirectional pins that are directly controlled by primary inputs, by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Example 1

The following example uses the set_bidi_gating command to insert test logic to control all bidi pins used for scan I/O via the SEN signal, and reports the gated bidi pin.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
add_scan_pins c1 bidi_in1/X blkB1/blkA/utri2/A -top io out1
set_system_mode analysis
report_dft_check -full

---------------------------------------------------------------------------
                        Bidi                Primitive   Control  Control
Tri-state         State Direction  Gating ID          Signal   Driver
---------------------------------------------------------------------------
/bidi_in1         OFF   IN          YES   20          SEN      /or2/Y
/blkB1/blkA/utri3 OFF   --          YES   15          SEN      /udff0/Q
/blkB1/blkA/utri2 ON    --          YES   17          SEN      /or2/Y
/blkB1/blkA/utri1 OFF   --          YES   16          SEN      /or2/Y
---------------------------------------------------------------------------
```

Example 2

The following example uses the force_gating switch to insert gating logic controlled by the TEN control signal on the enable line of /uio1 and reports the gated tri-state devices. /uio1 is a bidirectional device driving primary inout port dinout[1]; its enable signal is directly controlled by the primary input /io_control1.

```
set_bidi_gating on -control ten -direction input -top dinout[1] -force_gating
report_dft_check -tri



---------------------------------------------------------------------------
                  Bidi                  Primitive   Control  Control
```

```
Tri-state   State Direction   Gating   ID          Signal   Driver
-----------------------------------------------------------------------
/uioM       OFF   IN          YES      84          TEN      /udff20/QB
/uio3       ON    OUT         YES      77          SEN      /io_control
/uio2       OFF   IN          NO       76          SEN      /io_control
/uio1       OFF   IN          YES      75          TEN      /io_control1
/uio0       OFF   IN          NO       73          SEN      /io_control
-----------------------------------------------------------------------
```

Related Commands

report_dft_check
report_test_logic
report_control_signals
set_test_logic
set_tristate_gating

set_bist_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_bist_chain_test [None | *number_of_patterns*]

Description

Disables all capture clock activity for the specified number of patterns in the fault simulation during execution of the chain test.

By default, the tool runs a 32 pattern chain test and disables all capture clock activity for those 32 patterns, which means that the unload values will exactly match the load values during those patterns.

Hybrid TK/LBIST Flow

The tool automatically includes this command and literal in the generated LogicBIST fault simulation dofile—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- None

  An optional literal that specifies that the tool should not disable the capture clock activity since there is no chain test being ran.

- *number_of_patterns*

  An optional integer that specifies for the tool to disable all capture clock activity for the number of chain test patterns that are to be driven through the scan chains, which is 32 by default.

Related Commands

read_patterns

set_random_patterns

set_bist_debug

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_bist_debug **-None** | [***LFSR_name***… {***pattern_number*** | **All**}]

Hybrid TK/LBIST Flow Usage

set_bist_debug **-None** | [***LFSR_name***… {***pattern_number*** | **All**}] |
[**-decompressor** [*block_name*] {***pattern_number*** | **All**}]

Description

Sets up a trace of LFSR(s) value during a pattern's shift cycles.

To facilitate signature mismatch debugging and other diagnostics, the set_bist_debug command enables the tracing of all specified MISR and PRPG values associated with a given pattern, or all BIST patterns. If low-power hardware is enabled in the hybrid TK/LBIST flow, the low-power hold register value is traced along with its corresponding PRPG.

Arguments

- **-None**

  A switch that clears the list of PRPGs and MISRs the tool is currently monitoring.

- ***LFSR_name***

  A repeatable string that specifies the name of the MISRs and PRPGs whose values the tool will trace. To list all MISRs and PRPGs in the design, use the report_lfsrs command.

Of the next two arguments, you must choose one.

- ***pattern_number***

  An integer that specifies the pattern number for which the MISR or decompressor values will be traced.

- **All**

  A literal that specifies to write out the value of the specified MISRs and PRPGs only at the end of the pattern, for every pattern.

Hybrid TK/LBIST Flow Arguments

- **-decompressor**

  A required switch that takes an optional *block_name* to specify that decompressor values for this block should be reported. (The decompressor is acting as a PRPG during LogicBIST.) When using a single EDT module, the *block_name* is optional but is required in the modular EDT flow. If you do not specify *block_name*, the tool reports decompressor values for all blocks.

- *block_name*

  An optional string that specifies the name of the EDT block containing the decompressor to evaluate. The block_name must be one you defined previously using the add_edt_blocks command.

Example 1

The following example initiates a trace on the values of both the MISR and the PRPG for the pattern, 61, and shows the resulting transcript:

```
//  command: set_bist_debug misr prpg 61

//  command: simulate_patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 5732  #faults = 418
//  system mode = fault simulation    pattern source = 63 BIST patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test     process
//  simulated  coverage  in list  detected   patterns  patterns   CPU time
//  begin bist patterns: capture clock = /clk, observe point = MASTER
begin debug data for PRPGs pattern = 61
   PRPG         prpg    0   1000001111111010
   PRPG         prpg    1   0100000111111101
   PRPG         prpg    2   1001010011111110
   PRPG         prpg    3   0100101001111111
   .
   .
   .
   PRPG         prpg   14   0101011110010010
   PRPG         prpg   15   0010101111001001
   PRPG         prpg   16   1010000111100100
//   63         88.68%      387      31      5      0      0.04 sec
begin debug data for MISRs pattern = 61
   MISR         misr    1   1110110100100111
   MISR         misr    2   0111011010111110
   MISR         misr    3   0011101100111111
   MISR         misr    4   0001110111111111
   .
   .
   .
   MISR         misr   16   1100100000101111
   MISR         misr   17   1100100000101111
```

Example 2

The following example initiates a trace on the values of the PRPG and MISR for all patterns:

```
//  command: set_bist_debug misr prpg all

//  command: simulate_patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 5912  #faults = 513
//  system mode = fault simulation    pattern source = 63 BIST patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test     process
//  simulated  coverage  in list  detected   patterns  patterns   CPU time
//  begin bist patterns: capture clock = /clk, observe point = MASTER
```

```
   PRPG         prpg    0   0001001000001001
   PRPG         prpg    1   0011001000011010
   PRPG         prpg    2   0010111111001111
   PRPG         prpg    3   0110011101111011
   .
   .
   .
   PRPG         prpg   29   0001101111101000
   PRPG         prpg   30   0100010010110000
   PRPG         prpg   31   1110101100101111
   MISR         misr    0   0101010100001101
   MISR         misr    1   1101011101111111
   MISR         misr    2   1101010001101001
   MISR         misr    3   1110000010111001
   .
   .
   .
   MISR         misr   29   0010001111001110
   MISR         misr   30   1111000010000110
   MISR         misr   31   1111001001110101
   PRPG         prpg   32   1000110000011000
   PRPG         prpg   33   0101010011111101
   PRPG         prpg   34   0100001010111010
   .
   .
   .
   PRPG         prpg   60   1000001111111010
   PRPG         prpg   61   1010000111100100
   PRPG         prpg   62   0000000110101110
//   63         86.05%      488      25      3      0      0.10 sec
   MISR         misr   32   1000101100100001
   MISR         misr   33   0010101011010011
   MISR         misr   34   1001100110111101
   .
   .
   .
   MISR         misr   60   1011110010101001
   MISR         misr   61   0011100101011010
   MISR         misr   62   0001111111111100
```

Example 3

The following example initiates the trace on the values of the decompressor, which is used as a PRPG in the hybrid TK/LBIST flow, for pattern 4 and shows the resulting transcript. In this example, low power is enabled.

```
// command: set bist debug -decompressor 4

begin debug data for DECOMPRESSORs pattern = 4
  DECOMP  piccpu_edt_i.lfsm_vec   0  1001110010011001010100110010010
  LP      piccpu_edt_i.lfsm_vec   0  0100110000000110010010001100110
  DECOMP  piccpu_edt_i.lfsm_vec   1  0100111001001000101010011001001
  LP      piccpu_edt_i.lfsm_vec   1  0100110000000110010010001100110
  DECOMP  piccpu_edt_i.lfsm_vec   2  1010011101100100010101001100100
  LP      piccpu_edt_i.lfsm_vec   2  0100110000000110010010001100110
  DECOMP  piccpu_edt_i.lfsm_vec   3  0101011110110110001010100110010
  LP      piccpu_edt_i.lfsm_vec   3  0100110000000110010010001100110
  DECOMP  piccpu_edt_i.lfsm_vec   4  0010111110011011000101010011001
  LP      piccpu_edt_i.lfsm_vec   4  0100110000000110010010001100110
  DECOMP  piccpu_edt_i.lfsm_vec   5  1001011111001001100010101001100
```

```
    LP      piccpu_edt_i.lfsm_vec   5  01001100000001100100100011001 10
    DECOMP  piccpu_edt_i.lfsm_vec   6  01001011101001001100010101001 10
    LP      piccpu_edt_i.lfsm_vec   6  01001100000001100100100011001 10
  .
  .
  .
```

Related Commands

set_bist_trace

add_lfsrs

report_lfsrs

set_bist_trace

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_bist_trace [-Nolfsr | {-Lfsr *filename* [-Replace] [-Interval *interval*] [-Offset *offset*]}]

Hybrid TK/LBIST Flow Usage

set_bist_trace [-prpg_data_file *filename* -misr_data_file *filename*]
{[-prpg_pload_file *filename* -misr_pload_file *filename*] [-range *beg_int end_int*]}
[-low_power] [-replace]

Description

Enables the tracing of PRPG and MISR values associated with each BIST pattern to facilitate signature mismatch debugging and other diagnostics.

The traced values are logged in *filename* in the following format:

```
0 PRPG prpg 1111111111111111
0 MISR misr 11111111111111111111
1 PRPG prpg 1001010001011000
2 PRPG prpg 0101111001100101
…
32 PRPG prpg 1100100000000110
1 MISR misr 11100111011000010110
2 MISR misr 11001110011100001110
3 MISR misr 01011001100111011001
…
32 MISR misr 01011001100111011001
32 PRPG prpg 1100100000000110
…
```

The first element on each line is the pattern number, followed by the simulated element type (MISR or PRPG). The third element is the name of the simulated element. The names shown in the previous example are the default names assigned by the add_lfsrs commands included in the BIST-Controller-phase-generated fault simulation driver. You can edit the driver file to rename the elements.

Note

Do not modify or delete any header information in *filename*. Microcoded test bench instructions are generated based on information in this header.

The sequence of activities that occur within the simulation of the BIST controller determines the order of the lines in *filename*.

If you issue multiple set_bist_trace commands before the simulate_patterns command, only the last one issued is executed.

To reduce the volume of data written to *filename*, use the -Interval and -Offset switches. See the Example 1 section for this command.

Arguments

- -Nolfsr

  An optional switch that disables LFSR value tracing. This is the default.

- -Lfsr *filename*

  An optional switch and string pair that enables tracing of PRPG and MISR values during a BIST run and writes those values to the *filename* file.

- -Replace

  An optional switch that replaces the contents of *filename* if that file already exists.

- -Interval *interval*

  An optional switch and integer pair that writes the PRPG and MISR values to *filename* for the patterns beginning at 0 and separated by the interval given by the *interval* integer. The values for the first and last patterns are always written.

- -Offset *offset*

  An optional switch and integer pair that writes the PRPG and MISR values to *filename* beginning at the pattern number given by the *offset* integer. The values for the first and last patterns are always written.

Hybrid TK/LBIST Flow Arguments

- -prpg_data_file *filename* -misr_data_file *filename*

  Two optional switches and string pairs that enable tracing of the PRPG and MISR values for all patterns during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow.

- -prpg_pload_file *filename* -misr_pload_file *filename*

  Two optional switches and string pairs that enable tracing for a few important clock cycles of the PRPG and MISR values during a BIST run and write those values to the respective *filename* file. The files are written in a format compatible with the LV Flow and are used for parallel load simulations.

- -range *beg_int end_int*

  An optional switch and integer pair that defines a range of patterns to be written out to the PRPG/MISR files when using the -prpg_pload_file and -misr_pload_file options. By default, the full pattern range for a BIST run is written out.

- -low_power

  An optional switch that defines whether the hybrid TK/LBIST flow should have the low-power values written out to the PRPG file when using the -prpg_data_file and -prpg_pload_file options.

- -replace

  An optional switch that replaces the contents of *filename* if that file already exists.

Example 1

The following example writes traced values to the *lfsr.trace* file, but only writes the values for every 256th pattern to conserve disk space.

```
set_bist_trace -lfsr lfsr.trace -replace -interval 256
```

In the preceding example, if you discover a problem begins after pattern 1530, you might narrow the search by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1530 -interval 16
```

and then finish by using the following command:

```
set_bist_trace -lfsr lfsr.trace -replace -offset 1594
```

Example 2 (Hybrid TK/LBIST Flow)

The following example writes out PRPG and MISR data files, with low-power values:

```
set_bist_trace -prpg_data_file LV_WORKDIR/TOP.prpg_data_lbist \
    -misr_data_file LV_WORKDIR/TOP.misr_data_lbist \
    -low_power \
    -replace
```

Example 3 (Hybrid TK/LBIST Flow)

The following example writes out the PRPG and MISR parallel load data files, with low-power values, for the first 256 patterns:

```
set_bist_trace -prpg_pload_file LV_WORKDIR/TOP.prpg_pload_lbist \
    -misr_pload_file LV_WORKDIR/TOP.misr_pload_lbist \
    -low_power \
    -range 0 255 \
    -replace
```

Related Command

set_bist_debug

set_bus_handling

Context: dft -edt, patterns -scan

Mode: analysis

Usage

set_bus_handling {**Pass** | **Fail** | **Abort**} {**-All** | **-Pass** | **-Fail** | **-Bidi** | **-ABort** | [-Instances *instance_name*...] | ***bus_gate_id#***... | ***net_pathname...***}

Description

Specifies the bus contention results that you desire for the identified buses.

The set_bus_handling command preassigns the contention check handling result that you desire during simulation for the buses that you specify. Upon invocation, the tool automatically calculates the bus contention handling as documented under the set_contention_check command description. The tool rejects (from the internal test pattern set) ATPG-generated patterns that can cause bus contention. The set_bus_handling command lets you override the automatic contention calculations, thereby changing whether or not the tool performs a simulator-based check using such a pattern.

The tool resets the bus contention handling back to the automatically calculated value whenever you make a change to the modeling that requires the tool to perform a complete re-analysis of the contention mutual exclusivity (such as changing the net resolution or the pin constraints).

Caution

Overriding the automatically calculated contention check handling results can cause trouble downstream if there is a problem with the design that required modification due to the bus contention.

Arguments

- **Pass**

  A literal that specifies for the tool to not perform bus contention evaluations on the buses that you identify and to treat them as if they had passed. This allows the tool to retain patterns that it would otherwise reject due to a contention check failure.

- **Fail**

  A literal that specifies for the tool to treat the buses that you identify as if they had failed the bus contention evaluations. This causes the tool to reject patterns that it would otherwise retain due to passing the contention check.

- **Abort**

  A literal that specifies the bus aborted the bus contention evaluations before determining whether the bus passed or failed. This can be used with the analyze_bus -Drc_check command to verify ATPG constraints which you have added to correct bus failures.

- **-All**

  A switch that specifies for the tool to change all buses to a specified state reset.

- **-Pass**

  A switch that specifies for the tool to change to specified state for the buses that have previously failed the E10 DRC rule.

- **-Fail**

  A switch that specifies for the tool to change to specified state for the buses that have previously passed the E10 DRC rule.

- **-Bidi**

  A switch that specifies for the tool to change to specified state for bidi buses identified by the E10 DRC rule.

- **-ABort**

  A switch that specifies for the tool to change to specified state for the buses that have previously aborted the E10 DRC rule.

- -Instances *instance_name*

  An optional switch and repeatable string that specifies the instances where the included buses are to be changed to the specified state.

- ***bus_gate_id#***

  A repeatable integer that specifies the gate identification numbers of the buses whose contention handling you want to override. If a bus is cascading, you must specify the dominant bus.

- ***net_pathname***

  A repeatable string that specifies a net name of the buses to be changed to a specified state.

Example

The following example turns the bus contention checking off, allowing the bus to pass the evaluations. However, this action can cause trouble in the future if there is a problem with the design that required modification due to the bus contention.

report_bus_data 321

```
/FA1/ha1/XOR1/OUT/ (321) handling=fail type=strong #Drivers=4
  Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
  BUS Drivers: 156(SW) 252(SW) 307(SW) 308(SW)
```

set_bus_handling pass 321
report_bus_data 321

```
/FA1/Ha1/Xor1/OUT/ (321) handling=pass type=strong #Drivers=4
  Learn Data:poss_X=yes, poss_Z=yes, poss_mult_drivers_on=yes
  BUS Drivers: 156(SW) 252SW) 307(SW) 308(SW)
```

Related Commands

report_bus_data

set_contention_check

set_bus_simulation

Context: dft -edt, patterns -scan

Mode: analysis

Usage

set_bus_simulation [Global | Local]

Description

Specifies whether the tool uses global or local bus simulation analysis.

This command simplifies typical back end verification flows by providing the option to turn off global analysis and use only the values that can be determined by inspecting the immediate input gates (tri-state devices or TSDs) driving a bus. Figure 7-34 shows an example of how the setting of this command effects pattern creation.

Figure 7-34. Global versus Local Bus Simulation Analysis Example

Global analysis assumes a consensus value (0 or 1) is driven onto the bus when one or more of the tri-state enables is X and all inputs to the tri-states are the same (either all 0 or all 1). This most closely matches real hardware behavior. Local analysis assumes an X is driven onto the bus if any of the tri-state enables is X, regardless of the values on the inputs to the tri-states. It is more pessimistic, but more closely matches typical Verilog UDP behavior.

This command is provided to resolve compatibility issues with tools that do not perform global analysis. For example, if your design includes instances of TSDs for gating signals onto a bus, and their behavior in a timing-based simulator is as shown on the right in the preceding figure, using global analysis for pattern creation can result in simulation mismatches when you perform timing-based verification of the test patterns. You can avoid these mismatches by using this command to specify local bus simulation analysis for pattern creation.

Tip: The report_environment command will display the analysis type currently in effect.

Arguments

- Global

  Specifies for the tool to perform global bus simulation analysis. This is the default.

- Local

  Specifies for the tool to perform local bus simulation analysis.

Related Commands

report_environment

set_bypass_chains

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

Mode: setup

Usage

set_bypass_chains {-**Bypass_chain_number** ***chain_number***}
[-Edt_chains {*chain_name*…}] [-Pins *bypass_input bypass_output*]

Description

Specifies a list of scan chains to connect in series to form a particular EDT bypass chain.

This command allows you to customize the configuration of the bypass chains; for example, to meet unique routing requirements.

If you use this command to customize the bypass chains, you must specify all the EDT bypass chains.

Arguments

- **-Bypass_chain_number** ***chain_number***

  A required switch and positive integer pair that identifies the EDT channel to connect to the specified scan chains to form an EDT bypass chain.

Note

When you specify the number of channels using the set_edt_options command -*Channels* switch, the channels are numbered consecutively starting at 1.

- -Edt_chains {chain_name…}

  An optional switch and repeatable string pair that specifies the internal scan chains to connect in series to form the specified bypass chain. The chains are connected from channel input to channel output in the order specified.

- -Pins *bypass_input bypass_output*

  An optional switch and double string that specifies the input and output pins for bypass chains that exist in the design netlist before EDT logic is generated.

Note

Scan chains and bypass chains must use the same input and output pins.

Example 1

The following example sets up the EDT logic for a design with eight scan chains named *chain1* through *chain8.* The design is setup with two EDT channels and chains 1, 5 and 6 are connected in series to channel 1 to form bypass chain 1 and the results display. Any number in the Bypass# column with *no connections* next to it represents an undefined bypass chain.

set_edt_options -channels 2
set_bypass_chains -bypass_chain 1 -edt_chains chain1 chain5 chain6
report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ----------
//    1        chain1, chain5, chain6
//    2        no connections
//
```

The following commands define the remaining bypass chain and displays the completed user-defined bypass configuration.

set_bypass_chains -bypass_chain 2 -edt_chains chain2 chain3 chain4 chain7 chain8
report_bypass_chains

```
//    Reporting user-defined bypass chains:
//
//    Bypass#  EDT chains
//    -------  ------
//    1        chain1, chain5, chain6
//    2        chain2, chain3, chain4, chain7, chain8
//
```

Example 2

The following example shows a dofile segment that sets up the scan chains and bypass chains for EDT logic and then shows the resulting EDT logic file.

```
// Internal scan chains
add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains  chain2 grp1 scan_in2 scan_out2
add_scan_chains  chain3 grp1 scan_in3 scan_out3
add_scan_chains  chain4 grp1 scan_in4 scan_out4
//
// Bypass scan chains
set_edt_options -channels 2 -bypass use_existing_bypass_chains
set_bypass_chains -bypass_chain 1 -pins scan_in1 scan_out1 -edt_chains \
chain1 chain2
set_bypass_chains -bypass_chain 2 -pins scan_in2 scan_out2 -edt_chains \
chain3 chain4
```

Muxes are added for only the internal scan pins that are also bypass chain pins as shown in the following EDT logic file.

```
module TOP_edt_bypass_logic (edt_bypass, edt_channels_in, edt_channels_out,
                             edt_scan_in, edt_scan_out, edt_bypass_in, edt_bypass_out);
   input        edt_bypass;
   input  [1:0] edt_channels_in;
   output [1:0] edt_channels_out;
   output [3:0] edt_scan_in;
   input  [3:0] edt_scan_out;
   input  [3:0] edt_bypass_in;
   input  [1:0] edt_bypass_out;


   assign edt_scan_in[0] = edt_bypass ? edt_channels_in[0] : edt_bypass_in[0];
   assign edt_scan_in[1] = edt_bypass ? edt_channels_in[1] : edt_bypass_in[1];
   assign edt_scan_in[2] = edt_bypass_in[2];
   assign edt_scan_in[3] = edt_bypass_in[3];

   assign edt_channels_out[0] = edt_bypass ? edt_scan_out[0] : edt_bypass_out[0];
   assign edt_channels_out[1] = edt_bypass ? edt_scan_out[1] : edt_bypass_out[1];
endmodule
```

Related Commands

report_bypass_chains
reset_bypass_chains
set_edt_options

set_capture_clock

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

Mode: setup, analysis

Usage

set_capture_clock [*primary_input_pin*] [-Atpg] | ANY]

Description

Specifies the capture clock for random pattern simulation or, optionally, for all ATPG pattern simulation.

This command affects the capture clock that is used during DRC simulation and may be useful for designs with certain types of test logic that must be initialized properly to pass scan chain tracing.

The set_capture_clock command performs slightly differently depending on which tool you are using.

The set_capture_clock command specifies the name of the capture clock the tool uses during random pattern simulation. You can specify the name of either a specific pin or a clock procedure in a test procedure file that identifies the pin. In either case, the pin must be a currently defined clock pin. Also, the capture clock that you specify cannot have a pin constraint.

If you do not specify a capture clock with this command, the tool sets the capture clock to none. If there is no capture clock and there is only one clock in the circuit that is not a set or reset line, the tool sets that clock as the capture clock during rules checking and displays a warning message that identifies the capture clock.

You can use the report_environment command to list the capture clock and the report_clocks command to identify the current list of clocks.

Note

This command applies only to random pattern generation unless you include the -Atpg switch; then it applies to all scan patterns created during the ATPG process.

The set_capture_clock command specifies a particular clock as the capture clock during random pattern generation or, optionally, ATPG. For random pattern generation alone, the command takes effect only if you previously issued a “simulate_patterns -source random” command.

When you have specified a capture clock with this command, the tool may pulse other defined clocks, but it will not use them for capture. You can specify the capture clock by providing the name of a specific pin in a test procedure file that identifies a currently defined clock pin. The capture clock that you specify cannot have a pin constraint.

You can undo a previously specified capture clock choice by issuing this command without any arguments. This will make all clocks available for clocking and capture. If there is no specified

capture clock and there is only one clock in the circuit that is not a set or reset line, the tool sets that clock as the capture clock during rules checking and displays a message that identifies the capture clock.

Note

During ATPG, the capture clock specified with this command is ignored if one or more named capture procedures are available and their use is enabled with "set_capture_procedure on" (the default).

Arguments

- *primary_input_pin*

  An optional string that specifies the name of the primary input pin you want to assign as the capture clock.

- -Atpg

  An optional switch that directs the tool to use the specified capture clock as the only capture clock for all patterns created during the ATPG process, not just random patterns. This option also forces Tessent FastScan to pulse the capture clock only once per capture cycle per pattern, causing the tool to generate only combinational patterns (no RAM sequential, clock sequential, or chain test patterns). If specified in Setup mode, this switch turns on some additional design rule checks (DRCs).

  When using this switch in Setup mode, all named capture procedures (NCPs) are disabled by default when entering a non-Setup mode. In addition, the create_patterns command can then create only patterns with the specified capture clock. After entering a non-Setup mode, you can manually enable NCPs by using the set_capture_procedure On command. The tool then checks to ensure that each of the NCPs pulse the capture clock once and only once. The tool does not enable those NCPs that don't satisfy this clock sequence.

Note

When using Tessent TestKompress (EDT On), you must specify this switch to enable the command to work properly with EDT.

- ANY (Tessent FastScan and Tessent TestKompress only)

  An optional string that specifies any clock can be a capture clock. This is the default if *primary_input_pin* is not specified.

Example

The following example specifies a capture clock:

```
add_clocks 1 clock1
set_capture_clock clock1
set_system_mode analysis
set_random_patterns 612
```

Related Commands

add_clocks
delete_clocks
report_clocks
simulate_patterns
report_environment
set_capture_procedures
read_patterns

set_capture_handling

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

Mode: setup, analysis

Usage

set_capture_handling {**-Ls** {**Old** | **New** | **X**} **-Te** {**Old** | **New** | **X**}}

Description

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

Specifies how the tool globally handles the data capture of state elements that have C3 and C4 rule violations.

The set_capture_handling command gives you some ability to globally change how the tool simulates data capture in the presence of C3 and C4 clock rules violations. C3 and C4 clock rules checks ensure that a clock line cannot capture data affected by the clock, and that any data the clock does capture does not affect the clock line itself. The tool does not normally allow C3 or C4 data capturing violations during simulation.

For information on the C3 and C4 rules, refer to the "Clock Rules (C Rules)" section.

You can use this command to set the global data capture behavior for level sensitive (-Ls) or trailing edge (-Te) devices. For each device type, you can specify for the tool to simulate old, new, or X data at the source points. You can override the global data capture handling for individual state elements by using the add_capture_handling command.

Arguments

- **-Ls Old | New | X**

  A switch and literal pair that specifies how you want the tool to handle the data capturing of level-sensitive state elements. The literal choices are as follows:

  Old — A literal that specifies for the tool to determine the output value of a level-sensitive source state element by using the data that existed prior to the current clock cycle. The tool then passes this value to the state element's sink state elements. This is the default behavior of the tool upon invocation.

  New — A literal that specifies for the tool to determine the output value of a level sensitive source state element by using the data from the current clock cycle. The tool then passes this value to the state element's sink state elements. The tool limits the scope of this capture handling effect to the circuitry between the source and sink points. The tool will not propagate the newly-captured effect past the sink point.

X — A literal that specifies for the tool to determine the output value of a level-sensitive source state element by using the data from the current clock cycle unless the previous values are different from the current values. If the values differ, the tool passes an unknown (X) value to the state element's sink state elements.

- **-Te Old | New | X**

  A switch and literal pair that specifies how you want the tool to handle the data capturing of trailing edge sensitive state elements. The literal choices are as follows:

  Old — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data that existed prior to the current clock cycle. The tool then passes this value to the state element's sink state elements. This is the default behavior of the tool upon invocation.

  New — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data from the current clock cycle. The tool then passes this value to the state element's sink state elements. The tool limits the scope of this capture handling effect to the circuitry between the source and sink points. The tool will not propagate the newly-captured effect past the sink point.

  X — A literal that specifies for the tool to determine the output value of a trailing edge sensitive source state element by using the data from the current clock cycle unless the previous values are different from the current values. If the values differ, the tool passes an unknown (X) value to the state element's sink state elements.

Example

The following example changes the data capture handling for all trailing edge state elements that have C3 and C4 rule violations:

```
set_capture_handling -te new
```

Related Commands

add_capture_handling
delete_capture_handling
report_capture_handling

set_capture_procedures

Context: dft -edt, patterns -scan

Mode: analysis

Usage

set_capture_procedures {ON | OFf} {-All | -Created_capture_procedure | -Loaded_capture_procedure | [*procedure_name*…]}

Description

Enables or disables the tool's use of a specified subset of named capture procedures when generating test patterns.

By default, the tool uses all named capture procedures defined in the test procedure file, except any that fail to pass DRC. The tool never uses capture procedures that fail DRC.

This command is useful when you want to use a subset of the defined capture procedures for certain types of faults, because the remaining capture procedures may be unable to detect the faults.

Note

The set_capture_procedures command applies to test generation only, not to fault simulation. If you generate test patterns using "set_capture_procedure off", for example, fault simulation will still simulate those test patterns even if you issue a "set_capture_procedure on" command prior to fault simulation.

Arguments

- ON

  A literal that enables use of the specified named capture procedures during test generation. This is the default. If one or more named capture procedures is enabled with this option, the tool does not use the default capture procedure; it uses the named capture procedure(s) exclusively. In this case, if the conditions to test a fault require a particular sequence of clocks or cycles that is not defined in one of the named capture procedures, the fault will be untestable.

  Note

  As shown in the examples, use of the "on" option to specify a subset of the defined capture procedures must be preceded by "set_capture_procedure off -all".

- OFf

  A literal that disables use of the specified named capture procedures during test generation. If you use this option with "-all" to disable all named capture procedures, ATPG uses the default capture procedure, if present. Otherwise, it runs as if there are no capture procedures defined in the test procedure file.

Note

Use of the “on” option to specify a subset of the defined capture procedures must be preceded by “set_capture_procedure off -all”.

- -All

  A switch that enables or disables use of all capture procedures defined in the test procedure file. This is the default.

- -Created_capture_procedure

  A switch that enables or disables use of only the named capture procedures internally created from the create_capture_procedures command.

- -Loaded_capture_procedure

  A switch that enables or disables use of only the named capture procedures loaded from the read_procfile command or the procedure file associated with the scan group definition.

- *procedure_name...*

  A repeatable string that enables or disables use of only the named capture procedures with the user-specified name.

Example 1

The following example loads a fault list, disables use of all capture procedures, enables just the capture procedure named c1_capture, then creates patterns:

```
read_faults clock1_fault_list
set_capture_procedures off -all
set_capture_procedures on c1_capture
create_patterns
```

Example 2

The following example creates named capture procedures from generated patterns, then disables use of loaded capture procedures:

```
create_patterns
create_capture_procedures -pattern internal
set_capture_procedures off -loaded_capture_procedure
```

Related Commands

create_capture_procedures
delete_capture_procedures
report_capture_procedures
set_drc_handling

set_cell_library_options

Context: unspecified, all contexts

Mode: setup

Usage

set_cell_library_options [-Array_delimiter {square | angle}] [-Prefer_cell_library { on | off }]

Description

Sets the specified options for library parsing and design elaboration.

When you issue the command without arguments, the command reports the current library options settings.

Arguments

- -Array_delimiter {square | angle}

  An optional switch and literal pair that specifies either square brackets “[ ]” or angle brackets “< >” as the default array delimiter. The default is square brackets.

- -Prefer_cell_library { on | off } (available only in contexts that operate in no_rtl mode)

  An optional switch and literal pair that controls the tool’s behavior when reading in a design that has both a Verilog module and a cell library model with the same name. By default in all contexts that operate in no_rtl mode, the tool chooses the cell library model. In contexts that operate in rtl mode, the tool always chooses the Verilog module. For information about the rtl and no_rtl modes, refer to the set_context command description.

  When you specify “off” before issuing the set_current_design command, the tool chooses the Verilog module.

Example

The following command specifies that the default array delimiter is angle brackets:

```
set_cell_library_options -array_delimiter angle
```

Related Commands

read_verilog
delete_design
set_design_include_directories
set_current_design
set_design_macros
read_cell_library
delete_cell_library

set_cell_model_mapping

Context: dft -scan

Mode: setup, analysis

Prerequisites: You can only override the mapping for scan models of the same scan type. For example, a mux-DFF scan model can only be replaced by another mux-DFF scan model.

Usage

set_cell_model_mapping **-New_model *scan_model_name***
[-MODUle *module_name…* | -Instance {*pathname…* | *instance_expression…*} |
-File *filename*] [-MODEl *model_name…*] [-Output *scan_ouput_pin_name*]

Description

Overrides the non-scan to scan model mapping defined by the tool.

You can change the scan model for an individual instance, all instances under a hierarchical instance, all instances in all occurrences of a module in the design, or all occurrences of the model in the entire design. Additionally, you can change the scan output pin of the scan model in the same manner.

Note

Use this command for changing the existing non-scan to scan model mapping, as defined in the library, and *not* to define the mapping.

Refer to "Defining Scan Cell and Scan Output Mapping" in the *Tessent Scan and ATPG User's Manual* for conceptual information on this topic.

Arguments

- **-New_model *scan_model_name***

  A required switch and string that specify a new library model for the specified design instances.

- -MODUle module_name… | -Instance {pathname… | instance_expression…} |
  -File filename]

  An optional switch and string or repeatable string that specify the design instances whose models should be mapped to the library model specified by the -New_model switch. If this switch is omitted, all instances of the model are specified with the -Model option.

  -Module — Assigns the library model specified by -New_model to either all compatible design instances within the module module_name or to just the design instances of the model specified with the -Model option.

  -Instance — Specifies the pathname(s) of either a hierarchical or library instance(s).

For a hierarchical instance, assigns the library model specified by -New_model to either all compatible design instances within this hierarchical instance or to just the design instances of the model specified with the -Model model_name option.

For a library instance, maps either all compatible instances to the specified -New_model library model or to just the instances of the model specified with the -Model model_name option. The repeatable string can be in the form of an absolute instance pathname or a pathname in regular expression form.

-File — Specifies the name of the file containing pathname(s) of instances that should be mapped to the -New_model model, if compatible.

- -MODEl *model_name*

  An optional switch and string that specify the name of an existing model whose design instances are to be mapped to the model specified by -*New_model*. The specified model can be either a non-scan or a scan model. However, the following assumptions are made:

  Non-scan models — The new scan model is appropriate for the non-scan model.

  Scan model — The old and new scan models are appropriate for at least one common non-scan model.

- -Output *scan_ouput_pin_name*

  An optional switch and string pair that specify the name of the scan output pin to use in place of the the tool-defined scan output pin. The pin/port must have been declared as a scan-out port in the scan_definition section of -New_model.

Related Commands

None

set_chain_test

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

Available when EDT is Off

set_chain_test [-SUppress_capture_cycle {OFf | ON}]
[{ -SEquence *bit_string* | OFf} ... [-APpend | -REplace] }]

Available when EDT is On

set_chain_test [-SUppress_capture_cycle {OFf | ON}]
[{ -SEquence { *bit_string* | OFf } ... [-APpend | -REplace] }]
[-Type {All | Nomask | Mask [XOr_only | 1Hot_only]}]
[-NOMASK_First {ON | OFf}]

Description

Specifies binary sequences for chain test.

By default, chain test is limited to one or two fixed sequence patterns. For example, if the minimum scan chain length is more than three, a single chain test sequence comprised of repeated *0011* is used.

Use this command to add one or more custom sequences to the current chain test or replace the current chain test sequences with all new sequences.

Arguments

- -SUppress_capture_cycle {OFf | ON}

  An optional switch and literal pair that suppresses the capture cycle from the chain test patterns. In the capture cycles, constrained pins are forced to their constrained values and cell constraints are justified.

  By default, chain test patterns contain a capture cycle when there is no C1 violation. In case DRC detects C1 violations on the level-sensitive ports or set/reset ports, the tool automatically suppresses the capture cycle for the chain test patterns in order to avoid those C1-violated cells to be disturbed and prevent simulation mismatches. You can manually change the default setting (set_chain_test -Suppress_capture_cycle Off) to force the chain test patterns to include the capture cycle. Under this situation, the tool masks out the scan cells with a C1 violation on either synchronous port or set/reset ports to prevent mismatches because the C1-violated cells on a synchronous or set/reset port may cause the loading value to be disturbed in the capture cycle. Note that this masking effect could impact the diagnosability of the chain test failure but should not impact the overall test quality.

  This switch also suppresses the capture cycle of the first pattern (edt_setup) before chain and scan test patterns when the low-power EDT decompressor or input channel pipelining is in use. For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

- -SEquence *bit_string* | OFF (**Available when EDT is Off**)

  A optional, repeatable literal and string pair that specifies a binary sequence to shift into the scan chains, starting with the most significant bit. If the length of a specified bit string sequence is shorter than a scan chain, the specified sequence is repeated to fill the rest of the scan chain. If the specified sequence is longer than the scan chain, the extra less significant bits are truncated.

  Specifying OFF disables custom sequences. This allows you to revert back to using the default chain test.

- -APpend **(Available when EDT is Off)**

  An optional switch that specifies to add the specified sequences to the tool's current chain test sequences.

- -REplace (**Available when EDT is Off)**

  An optional switch that specifies to replace the tool's current chain test sequences with the specified bit sequences. This is the default.

- -Type All | Nomask | Mask [XOr_only | 1Hot_only] (**Available when EDT is On**)

  An optional switch and literal that specifies the type of EDT chain test patterns that should be saved to minimize effort during simulation run time. The chain test patterns are saved when the write_patterns command is issued. The possible literal values for -*Type* are:

  All — Saves chain test patterns with all types of masking. This option includes all non-masking and masking patterns as well as patterns to test all unused decoder values when all chain patterns are selected. As a result, the total number of patterns returned by this option may be greater than the sum of all non-masking and masking patterns. For more information on unused patterns, see "EDT Logic and Chain Testing" in the *Tessent TestKompress User's Manual*.

  Nomask — Only saves the non-masking chain test patterns.

Note

If you are *not* using the low-power decompressor and you are constrained by the number of patterns you can simulate, you can run one non-masking pattern to achieve minimum coverage, as shown here:

```
set_chain_test -type nomask
write_patterns pattern_filename -pattern_sets chain -serial -end 0
```

If you *are* using the low-power decompressor, it is safest to run all non-masking patterns so that every chain is tested, as shown here:

```
set_chain_test -type nomask
write_patterns pattern_filename  -pattern_sets chain -serial
```

Mask — Only saves the masking chain pattern.

XOr_only — For the Xpress compactor, saves only flexible masking chain patterns. Note: This does not apply to the basic compactor.

1Hot_only — Saves only 1-hot masking chain patterns.

Note

If you do not save 1-hot patterns (*Nomask* or *Mask [XOr_only]* options), Tessent Diagnosis may not have enough information to run chain diagnosis successfully. Without 1-hot patterns, chains in a compactor are scanned out together and the tool cannot identify the specific chain that contains the error. If you do not want to save 1-hot patterns but still want to run chain diagnosis, you must communicate to Tessent Diagnosis the failing chains using the following commands:

```
read_patterns scan_patterns
diagnose_failures <failure_file> -faulty_chain <faulty_chain_name>
        <faulty_chain_type>
```

This approach assumes internal failing chain information was reported through another method. For more information, see the read_patterns and diagnose_failures commands.

For more information on masking patterns, see "Understanding Scan Chain Masking in the Compactor" in the *Tessent TeskKompress User's Manual*.

- -NOMASK_First {ON | OFf} (**Available when EDT is On**)

  An optional switch that specifies whether to place EDT non-masking chain patterns first. The default is OFf which means that 1-Hot masking chain patterns are ordered first. The -Nomask_First ON option is only valid with -Type All— the option is ignored if used with -Type Mask.

set_checkpointing_options

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_checkpointing_options [OFf | ON { [-PATTERN_FIle *filename*]
[-PATTERN_FOrmat {Ascii | Binary}] [-Faultlist_file *filename*] [-Replace]
[-Period *period*] }]

Description

Enables checkpointing and specifies the associated options.

Checkpointing is when the tool automatically saves test patterns at regular periods, referred to as checkpoints, throughout the pattern creation process. This is useful when ATPG takes a long time and there is a possibility it could be interrupted accidentally.

The set_checkpointing_options command specifies the filename and time period in which the tool writes test patterns during test pattern generation.

You can use the -Faultlist_file option to save the fault list. In the case of an interruption, if you have saved the generated patterns, you can use this fault list to avoid the resimulation step. When you specify the -Faultlist_file option, the tool saves the checkpointed fault list, including the fault class and sub-class of each fault, to the specified file. Aborted faults are also written to the file.

You can use the set_checkpointing_options command multiple times within a session and each time modify only the options you want to change. The tool retains your unmodified settings from the previous commands.

Note

Although you can issue the set_checkpointing_options command in any mode, the tool only writes out a checkpoint file in analysis mode with the set_checkpointing_options command set to On.

For additional information on checkpointing, refer to "Setting Up for Checkpointing" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- OFf

  A literal that specifies for the tool not to use the checkpointing functionality during test pattern generation. Patterns are not written to any file. This is the invocation default.

- ON

  A literal that specifies for the tool to use the checkpointing functionality. The tool writes test patterns that it generates to the file that you specified with the set_checkpointing_options command.

- -PATTERN_FIle *filename*

  An optional switch and string pair that specifies the name of the file into which you want to write the test patterns during test pattern generation.

- -PATTERN_FOrmat {Ascii | Binary}

  An optional switch and literal pair that allows the pattern files to be saved in binary or ASCII format. Binary is the default format.

- -Faultlist_file *filename*

  An optional switch and string pair that allows the fault list to be saved. Tessent FastScan also saves a second file with the same name (but with the added suffix ".protected") that contains only the information for protected faults. The tool saves protected fault information at the end of a time period only if the protected faults have changed since the last save.

- -Replace

  An optional switch that forces the tool to overwrite the file if it already exists. If you use the -Replace option and the tool does not create any new test patterns, the tool does not update the file.

- -Period *period*

  An optional switch and integer pair that specifies the number of minutes between each write of the test patterns. The default is 100 minutes.

Example

The following example stores the generated test patterns every 120 minutes in a file called "my_checkpoint_file". After each 120-minute interval, the tool creates a new file until the ATPG process ends.

```
set_system_mode analysis
set_checkpointing_options on -pattern_file my_checkpoint_file -period 120
create_patterns
```

Related Commands

write_flat_model
write_patterns
create_patterns
write_faults

set_clock_controller_pins

Context: dft -logic_bist

Mode: setup

Usage

set_clock_controller_pins
{shift_clock | shift_clock_en | scan_en | lbist_en **| capture_procedure_index}**
{***internal_pin_name***… | -no_connection} [-append]

Description

Specifies the connection information for the clock controller pins.

This command is used with the Hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

If you issue this command multiple times with the same pin type, then the last specified pin name(s) are used. To specify multiple pin names corresponding to a pin type, specify all the names in the same command, or use the -append switch to add to the list of already specified pins.

The pins you specify with this command are connected to the single top-level LogicBIST controller. This command is global in context and not specific to the current EDT block.

Alternatively, you can specify the -no_connection switch instead of internal_pin_name in an RTL IP-creation flow (skeleton flow), where the clock controller is not yet available in the dummy design.

Note

The internal_pin_name string and -no_connection switch are mutually exclusive.

Arguments

- **shift_clock *internal_pin_name***

  A required literal and repeatable string that specifies the clock controller shift clock input. This pin is to be driven by the LogicBIST shift clock output from the controller in LogicBIST mode.

- **shift_clock_en *internal_pin_name***

  A required literal and repeatable string that specifies the clock controller shift clock enable input. This pin is to be driven by the LogicBIST shift clock enable output from the controller in LogicBIST mode.

- **scan_en *internal_pin_name***

  A required literal and repeatable string that specifies the clock controller scan enable input. This pin is to be driven by inverted LogicBIST capture enable output from the controller in LogicBIST mode.

- lbist_en ***internal_pin_name***

  An optional literal that specifies the LogicBIST enable pins. You can use this to specify the lbist_en pin on the clock-controller interface.

- **capture_procedure_index** ***internal_pin_name***

  A required literal that specifies the connections to the signal that identifies the NCP used for the current pattern. Then internal_pin_name string identifies the pins in the clock controller for the NCP index specified from MSB to LSB.

  A capture_procedure_index pin type has to be specified once per clock controller to add the list of NCP index pins for that clock controller. It is not possible to group the connections for the capture_procedure_index pin type for all clock controllers in a single command. The -append switch must be used to add to the pin specification for prior controllers.

- -no_connection

  An optional switch used instead of internal_pin_name.

  When the -no_connection switch is used, the design clock controller pin names need not be specified and no connections will be made. The pins are available at the LogicBIST IP boundary. This switch can be used in an RTL IP-creation flow, where the clock controller is not yet available in the dummy design.

- -append

  An optional switch used to append to a previous list of pin names, instead of replacing. Without the -append switch, repeat usage of the set_clock_controller_pins command overwrites the previous list of pins for the specified pin type.

Example 1

The following example specifies using controller/scan_en as the scan enable input pin on the clock-controller interface:

```
set_clock_controller_pins scan_en controller/scan_en
```

Example 2

In this example, assume that the clock controller is not available in the design. The following example specifies the clock controller's shift clock but does not make any connection to the clock-controller interface:

```
set_clock_controller_pins shift_clock -no_connection
```

Example 3

This example connects the lbist_en output of the logicBIST controller to the LogicBIST enable pin on multiple clock controllers. Note that the arguments should be specified in a list surrounded by braces.

```
set_clock_controller_pins lbist_en \
        {cc_clk1/lbist_mode cc_clk2/lbist_mode cc_clk3/lbist_mode}
```

Example 4

This example shows how to connect the NCP index pins of multiple clock controllers using the -append switch.

Note

You must specify a connection for each clock controller separately as the capture_procedure_index pin type requires a list of pin names be declared for each clock controller.

```
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk1/ncp*]
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk2/ncp*] -append
set_clock_controller_pins capture_procedure_index \
      [get_pins cc_clk3/ncp*] -append
```

Issuing a report_clock_controller_pins command will show the following:

```
//  command: report_clock_controller_pins
//
//  PinType                  Connection
//  -------                  ----------
//  LBIST enable             cc_clk1/lbist_mode cc_clk2/lbist_mode
//                           cc_clk3/lbist_mode
//  Capture procedure index  cc_clk1/ncp_index[1] cc_clk1/ncp_index[0]
//                           cc_clk2/ncp_index[1] cc_clk2/ncp_index[0]
//                           cc_clk3/ncp_index[1] cc_clk3/ncp_index[0]
//
```

Related Commands

report_clock_controller_pins

set_lbist_controller_options

set_clock_controls

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

Mode: setup, analysis

Usage

set_clock_controls {**ON** | **OFf**}
[-ENFORCE_off_conditions_outside_capture_if_always_capture_source { ON | OFf }]

Description

Enables and disables the clock control definitions in the test procedure file.

When enabled, the clock definitions are used for ATPG. Clock control definitions cannot be used simultaneously with Named Capture Procedures (NCPs).

For more information, see "Support for Internal Clock Control" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **ON**

  Required literal that enables clock control definitions. This is the default setting unless Named Capture Procedures (NCPs) are enabled in the test procedure file. When NCPs are enabled, the internal clocks are automatically disabled.

- **OFf**

  Required literal that disables clock control definitions. Unless the internal clocks are controlled by NCPs, turning clock control off leads the tool to treat the internal clocks as directly controllable, ignoring the constraints and condition bits imposed on ATPG by the clock control definitions. Any generated patterns are invalid since the clock controller requirements modeled using the clock control definitions are completely ignored by ATPG and simulation. You should turn off clock controls only for debug purposes, typically to determine if a fault is untestable because of the clock control constraints and if the fault is testable when those constraints are not present.

- -ENFORCE_off_conditions_outside_capture_if_always_capture_source { ON | OFf }

  An optional switch and keyword pair that enables turning off the following default behavior: when the source clock of a clock control is an always_capture clock, the tool uses condition bits to ensure that the internal clock is off in cycles beyond those generated by ATPG even if the source clock is pulsed beyond that point. This prevents disturbing the captured values.

  When generating patterns in the presence of internal clocks controlled by using clock control definitions, the tool must either decide whether to use condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern or assume that disabling the source clock will suppress further internal clock pulses. For example, consider that the clock control definition for a given internal clock specifies three condition bits: B1 to control the clock pulsing in the first cycle, B2 to control it pulsing in the second cycle, and B3 to control it pulsing in the third cycle. If ATPG generates a one-

cycle pattern that pulses this clock, the tool encodes B1=1. The tool then has the option of encoding B2=0 and B3=0, or leaving them unspecified to be random-filled.

If the clock control definition has no source clock defined or has a free-running clock, then the tool must encode those two additional condition bits to ensure that any pulses the OCC receives beyond the first pulse are suppressed and the internal clock only pulses once. On the other hand, if the clock control definition has a source clock that can be controlled by the tester and is now defined as always_capture or free_running, this source clock will only be pulsed once, and there is no need to waste encoding capacity to encode 00 into those two bits because the internal clock pulses only once regardless of the values loaded into B2 and B3.

The tool decides whether to use condition bits to suppress further internal clock pulses as follows:

1. If a clock control definition has no source clock, or its source clock(s) are free-running, the tool applies the condition bits needed to suppress additional internal clock pulses beyond the sequential depth of the generated pattern. You have no way to change this.
2. If generating retargetable patterns (“set_current_mode -type internal”), the tool applies the condition bits to suppress additional internal clock pulses. You cannot change this because the clocks at the core boundary typically do receive additional clock pulses when retargeted and merged with patterns for other cores.
3. If the source clock(s) defined in a clock control block are always_capture, the tool applies the condition bits to suppress additional internal clock pulses. You can override this behavior by using “set_clock_controls on -enforce_off_conditions_outside_capture_if_always_capture_source off.” This reduces the number of specified bits and may improve EDT compression. However, if you use this option, you must be sure that the source clock will not be pulsed additional times than ATPG has pulsed it. Do not use the “set_external_capture_options -fixed_cycles” or “set_external_capture_options -minimum_cycles” commands when writing your patterns, since those options pulse always_capture clocks when adding the padding cycles. Note that capture cycle padding is used for the type-3 LPCT controller, and as such, you should not use “set_clock_control on -enforce _off_conditions_outside_capture_if_always_capture_source off” in that case either.
4. If none of the previous conditions are satisfied and the source clock(s) are defined and completely under ATPG control with no constraints, then ATPG does not apply the condition bits to suppress additional internal clock pulses beyond the sequential depth of the generated pattern.

Example

The following example is a debugging flow that disables all clock control definitions in the test procedure file and generates test patterns:

```
set_clock_controls off
create_patterns
```

Related Commands

report_clock_controls

set_clock_gating

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_clock_gating {ON | OFf} |
{[-LIbrary_model *library_model_name*…]
[-Module *module_object*...] [-INStance *instance_object*...]
[-Port_to_connect port_name [-INVert]]
[-Clock_port *port_name*]
[-Driver pin_pathname [-Active {High | Low}]]}

Description

Specifies clock gating cells whose unconnected scan enable ports need to be connected to either the scan enable signal or a specified signal (pin).

The tool considers an unconnected port to be a port that is not connected to any net or is tied low (TIE0), tied high (TIE1), tied unknown (TIEX), or tied high impedance (TIEZ).

The tool does one of the following:

- Automatically checks if the scan enable port of each recognized clock gating instance is unconnected and, if unconnected, connects it to either the scan enable signal or the signal specified by the -Driver pin_pathname argument. This occurs when none of the following switches is specified: -Library_model, -Module, -Instance, or -Port_to_connect.
- Automatically checks if the specified port of each specified instance is unconnected and, if unconnected, connects it to either the scan enable signal or the signal specified by the -Driver pin_pathname argument.

Arguments

- ON

  A literal that enables clock gating.

- OFf

  A literal that turns off clock gating. Any previously specified clock gating commands are ignored as if they were never issued. This is the default.

Note

The ON or OFF switch cannot be used with any of the other switches; doing so will result in an Error.

- -LIbrary_model *library_model_name*

  An optional switch and a repeatable string that specify the library model name(s) of the clock gating cell instances whose unconnected port is to be connected to either the scan enable signal or to the specified signal.

- -Module *module_object*

  An optional switch and string pair that specify the netlist module name(s) of the clock gating cell instances whose unconnected port is to be connected to the specified signal. The module_object can be a single module name, a TCL list of module names, or a collection of module objects as returned by the get_modules command. This option requires both the -Port_to_connect and -Driver options to be specified as well. The tool only attempts to connect the specified port of the specified clock gating cell if it determines the port is really unconnected.

- -INstance *instance_object*

  A optional switch and string pair that specify the hierarchical instance pathname(s) of the clock gating cell instances whose unconnected port is to be connected to either the scan enable signal or to the specified signal. The instance_object can be a single instance pathname, a TCL list of instance pathnames, or a collection of instance objects as returned by the get_instances command.

- -Port_to_connect *port_name* [-INVert]

  An optional switch and a string pair that specifies the unconnected port of the clock gating cell to be connected to either the scan enable signal or to the specified signal. This option must be specified when the -Module switch is specified.

  -INVert — By default, the expected input value for the unconnected port is set to 1 during the shift cycle. This switch specifies that the expected input value is set to 0 during the shift cycle.

- -Clock_port *port_name*

  An optional switch and a string pair specifying the clock port of the clock gating cell. This option should be used when the clock gating device is to be connected to an enable signal associated with the clock domain.

- -Driver *pin_pathname* [-Active {<u>High</u> | Low}]

  An optional switch and literal pair that specifies the pin pathname for the signal driver that the unconnected port is to be connected to. The specified pin_pathname can be either a top-level port or an internal pin instance. When this option is not specified, the tool assumes the unconnected ports are to be connected to the scan enable signal. This option must be specified when the -Module switch is specified.

  -Active — An optional switch that indicates whether the specified signal is active Low or High. By default, the tool assumes the active state is High, unless it is an existing global signal whose active state is known.

Examples

The following example connects the unconnected scan enable ports of specified clock gating instances. The first instance is connected to the sen1 pin which drives the signal with the active state set to low. The next two instances are connected to the default scan enable signal, sen. The results of these commands are shown by the report_clock_gating command output:

```
set_scan_enable sen
set_clock_gating -instance clkg1/clkg1/clkgLA -driver sen1 -active low
set_clock_gating -instance {clkg3/clkg1/clkgLA clkg2/clkg1/clkgLA}
set_system_mode analysis

// Note: The following clock gating instances have unconnected ports that
will be connected to a scan enable signal.
---------------------------------------------------------------------------
Clock Gating            Unconnected       Signal
Instance                Port              Driver
---------------------------------------------------------------------------
clkg1/clkg1/clkgLA      SE                sen1
clkg2/clkg1/clkgLA      SE                sen
clkg3/clkg1/clkgLA      SE                sen
---------------------------------------------------------------------------
.......
insert_test_logic
report_clock_gating -instance [get_instances clkg* -below_instances
                                                          clkg*/clkg1]
---------------------------------------------------------------------------
Clock Gating            Unconnected       Signal
Instance                Port              Driver
---------------------------------------------------------------------------
clkg1/clkg1/clkgLA      SE                sen1
clkg2/clkg1/clkgLA      SE                sen
clkg3/clkg1/clkgLA      SE                sen
---------------------------------------------------------------------------
```

Related Topics

report_clock_gating

set_clock_gating_enable

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_clock_gating_enable ***enable_signal_driver_pathname*** [*primary_input*]
[-Active {High | Low}]
-Clock_domain *clock_signal_driver_pathname*…

Description

Associates clocks that control clock gating devices with enable signal drivers.

This command assume that the clocks controlling the clock gating devices (whose enable ports are to be automatically connected) were added to the design with the add_clocks command.

The tool connects the unconnected enable ports of clock gating devices to the enable signal associated with the controlling clocks of these clock gating devices. The clock gating devices, the enable ports of the clock gating devices, and their clock ports must be specified with the set_clock_gating command.

- ***enable_signal_driver_pathname***

  A required string that specifies either the top-level input port or an existing internal output pin to be connected to the unconnected enable port of a clock gating device controlled by the clock specified with the -Clock_domain switch. The enable_signal_driver_pathname argument can be either a top-level enable port or an internal instance pin.

  If an internal instance pin is specified, the tool attempts to locate its primary input driver by tracing back to a primary input via a simple path; that is, through only inverters or buffers. If automatic identification of the primary input driver fails, the value of the primary_input option is used, if provided. Identification of the primary input is needed to generate the ATPG dofile and test procedure files.

- primary_input

  An optional string that specifies a top-level enable port. This argument supplies a primary input/top-level port for an internal instance pin specified by the enable_signal_driver_pathname argument. The specified top-level port is used when generating the ATPG dofile and test procedure files. If the specified top-level port does not exist, the tool creates the port The specified top-level port must be a primary input port.

- -Active {High | Low}

  An optional switch and literal pair that specify whether the enable signal is active low or active high. The default setting is high.

- **-Clock_domain *clock_signal_driver_pathname…***

A required switch and a repeatable string that specifies the defined clock(s) controlling clock gating devices whose unconnected enable ports are to be connected to the specified enable signal's driver.

set_clock_off_simulation

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

Mode: setup, analysis

Usage

set_clock_off_simulation **ON** | **OFf**

Description

Enables or disables the simulation where all clock primary inputs are at their “off” value, other primary inputs have been forced to values, and state elements are at the values scanned in or resulting from capture in the previous cycle.

When simulating this event, the capture data for inputs is provided to leading edge triggered flip-flops.

Tip: If you use ATPG Expert (create_patterns), you do not need to issue this command for better handling of DRC violations. ATPG Expert automatically enables it when needed. You may get better results by letting ATPG Expert decide when to use this command except in the case of transparent latch handling.

Note that the set_clock_off_simulation command affects ATPG (subsequent create_patterns commands) but not logic simulation (subsequent simulate_patterns commands).

Arguments

- **ON**

  A literal that performs ATPG for all clock primary inputs at their “off” values.

- **OFf**

  A literal that performs ATPG for all clock primary inputs at their active values. This is the default.

Example

The following example enables clock_off simulation prior to creating patterns:

```
set_system_mode analysis
set_clock_off_simulation on
create_patterns
```

Related Commands

set_split_capture_cycle

set_clock_restriction

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

Mode: setup, analysis

Usage

set_clock_restriction { **Clock_po** | **ON** | **OFf** | **Domain_clock** [-Edge_interaction
| -Any_interaction] } [-PULSE_interacting_clocks { OFf | ON }]
[-MAX_SKEW { Half_cycle | One_cycle | Multiple_cycles }]
[-SAME_clocks_between_loads {OFf | ON}]
[-IGNORE_unobservable_cells {ON | OFf}]

Description

Specifies whether ATPG can create patterns with more than one active capture clock.

The set_clock_restriction command changes the default behavior of ATPG regarding the creation of test patterns that have more than one active clock line.

Note

Free-running clocks are excluded from clock restriction checking when you choose Clock_po or ON. For example, if you choose ON, and two free-running clocks are defined, then only one clock is allowed in addition to the free-running clocks.

If you choose Domain_clock, the tool forces free-running clocks to be compatible with other clocks.

Arguments

- **Clock_po**

  A literal that specifies for the ATPG process to create patterns independent of the clock restriction.

  If the ATPG creates a pattern that requires multiple active clocks but does not detect the fault at the clock PO, the ATPG rejects the pattern and displays a warning message at the end of the run indicating the number of rejected patterns. When you change the clock restriction to off, the tool resets the ATPG untestable faults to undetected-uncontrolled.

  Pulsing incompatible clocks simultaneously can cause simulation mismatch due to clock skew between them. Do not turn off the clock restriction. It is recommended to use “set_clock_restriction domain_clock” for the tool to identify the clocks that can be pulsed simultaneously during capture. To report the clock compatibility, use the command “report_clock_domains -Compatible_clocks -Details”.

  Allows only clock_po patterns to have multiple active clocks. Domain_clock allows more than just clock_po patterns to have multiple active clocks.

- **ON**

  A literal that specifies for the ATPG to create only patterns with, at most, a single active clock.

  The ATPG treats equivalent clocks as a single clock, however, it treats multiple active clocks as a conflict condition and performs an exhaustive search for conditions necessary to detect the fault with no more than one active clock. Faults which the ATPG detects at primary outputs (POs) that connect to clocks must also satisfy this condition. Faults detected at clock POs that require multiple active clocks for detection, require that you use the Clock_po option. You can accomplish this by setting the clock restriction to Clock_po and then re-running the ATPG.

  During the bus contention prevention analysis portion of the ATPG, Tessent FastScan and Tessent TestKompress turn off any clock pins that the ATPG does not require for fault detection.

- **OFf**

  A literal that specifies for the ATPG process to create patterns with as many clocks on as it requires to detect faults. Using this option may cause race conditions due to multiple active clocks. Prevent these race conditions by specifying the On argument.

  For Tessent FastScan and Tessent TestKompress — These tools concurrently apply any clocks they require for a given pattern. They display a message at the end of the ATPG run to indicate the number of patterns that had more than one active clock. When you change the clock restriction to off, these tools reset the ATPG untestable faults to undetected-uncontrolled.

- **Domain_clock**

  A literal that specifies for ATPG to create patterns with compatible clocks on. This enables the tool to generate patterns that pulse compatible scan chain capture or system clocks at the same time. Normally, only one scan chain capture or system clock is pulsed in a cycle. The tool treats two clocks as compatible if their domains are completely independent of each other; that is, they can be in their “on” state at the same time without causing clock skew problems or race conditions.

  The tool automatically analyzes circuit data from DRC to determine the compatible clocks. Therefore, if you use this option prior to DRC, the tool defers execution until DRC.

Note

The tool’s analysis of clock domains only considers Tie-0 or Tie-1 gates that are Tie-0 or Tie-1 in both combinational and sequential ATPG.

- -Edge_interaction

  An optional switch that, when the Domain_clock setting is in effect, specifies for the tool to determine if two clocks are compatible (C) or incompatible (I) based on their edge interactions, as summarized in Table 7-37. The table covers the case where a source

memory element is clocked by one clock, a sink memory element is clocked by a different clock, and the source and sink interact (the sink can capture data from the source).

Note

Take care when interpreting what the “C” and “I” mean: “C” means only that the tool treats the result of the particular combination as being unaffected by clock skew. “I” means the tool treats the result of the particular combination as susceptible to skew.

You can issue the set_clock_restriction command with only this switch, and the tool will retain whatever clock restriction setting (Clock_po, On, Off or Domain_clock) is currently in effect. This switch has no effect on the analysis used for ATPG unless Domain_clock is in effect; however, it does affect the output of the report_clock_domains command.

This is the default upon invocation of these tools.

Table 7-37. Default Clock Compatibility Summary at a Clock Domain Boundary

		Sink			
		LE[1]	**TE[2]**	**AH(TE)[3]**	**AL(LE)[4]**
Source	**LE[5]**	I	C	C	I
	TE[6]	C	I	I	C
	AH(LE)[7]	I	C	C	I
	AL(TE)[8]	C	I	I	C

1. LE = Edge sensitive element captures on PI clock’s leading edge
2. TE = Edge sensitive element captures on PI clock’s trailing edge
3. AH(TE) = Active high, level sensitive element captures on PI clock’s trailing edge
4. AL(LE) = Active low, level sensitive element captures on PI clock’s leading edge
5. LE = Edge sensitive element changes on PI clock’s leading edge
6. TE = Edge sensitive element changes on PI clock’s trailing edge
7. AH(LE) = Active high, level sensitive element changes on PI clock’s leading edge
8. AL(TE) = Active low, level sensitive element changes on PI clock’s trailing edge

- -Any_interaction

  An optional switch that, when the Domain_clock setting is in effect, specifies for the tool to do a more conservative analysis of the clocking relationship between interacting source and sink memory elements in different clock domains. If the source is clocked by one clock, the sink is clocked by a different clock, and the two elements interact (the sink can capture data from the source), this switch causes the tool to treat the two clocks as incompatible. In other words, all the conditions shown in Table 7-37 result in a determination of incompatibility.

You can issue the set_clock_restriction command with only this switch, and the tool will retain whatever clock restriction setting (Clock_po, On, Off or Domain_clock) is currently in effect. This switch has no effect on the analysis used for ATPG unless Domain_clock is in effect; however, it does affect the output of the report_clock_domains command.

- -PULSE_interacting_clocks { OFf | ON }

  An optional switch and literal pair that specifies whether the interacting clocks can be pulsed simultaneously during test. By default, the tool disables the pulsing of interacting clocks. The ON option allows pulsing pairs of interacting clocks during test generation. Note that this switch is effective only when you also specify the Domain_clock option.

  When this switch is ON, the tool masks some state elements located at the inter-clock domain in order to avoid simulation mismatches. Therefore, you will notice more X values at the state elements than when this switch is OFF. The behavior of cross-domain path masking depends on the setting of the max clock skew, as described below for the -Max_skew switch.

Note

The tools treats user-defined norace clocks (created with the add_norace_clocks command) as compatible clock domains and does not mask out cross-clock domain paths between norace clocks.

- -MAX_SKEW { Half_cycle | One_cycle | Multiple_cycles }

  An optional switch and literal pair that specifies the maximum skew between interacting clocks pulsed during capture. The maximum skew determines whether the observation state element should be masked or if fault simulation can give credit for detection. This option is effective only when the -Pulse_interacting_clocks option is enabled.

  The maximum skew settings impact fault simulation as follows:

  - Half_cycle — Only masks cross-clock domain paths of the same edge.
  - One_cycle — Masks cross-clock paths of the same clock edge, leading edge to trailing edge of the same cycle, and trailing edge to the leading edge of the next cycle. It allows the transition to be tested by launching at the leading (trailing) edge of one clock and capturing at the next cycle leading (trailing) edge of another clock.
  - Multiple_cycles — Masks all cross-clock domain paths. Tool assumes multiple clock cycles between all cross-clock domain paths and simulates them as asynchronous.

  The default maximum skew settings are as follows:

  - Static fault models (stuck-at and bridging) — Default setting is Half_cycle, although the default is One_cycle when you also use the -Any_interaction switch.
  - At-speed fault models (transition and path delay) — When the -same_clock_between_loads switch is disabled, interacting clocks may be pulsed in adjacent cycles. In this case, the default setting is One_cycle, which allows transition from one edge to the same edge in the next cycle. When the

-same_clocks_between_loads switch is enabled, interacting clocks are pulsed in every cycle and all clock interactions should be considered invalid. To reduce the chance of simulation mismatches, the default setting is Multiple_cycle.

You can use the -max_skew switch to overwrite the default clock skew to either prevent simulation mismatch (by specifying more clock skew than the default) or obtain better test coverage (by specifying less clock skew than the default). However, when changing the max skew to be less than the default, it must match with the design clock timing; otherwise, there is the possibility of simulation mismatch.

- -SAME_clocks_between_loads OFf | ON

  An optional switch and literal pair that specifies whether to use the same clock in every cycle within a load/unload interval during pattern creation. This switch is useful for at-speed test generation when you want each pattern to test for delay faults in only one clock domain.

  Generally, each test pattern created with this switch enabled will pulse the *same* clock in every test cycle that pulses a clock. Multi-load patterns are an exception: for a multi-load pattern, the switch applies to each load/unload interval independently. Therefore, the clock used in a particular interval might not be the clock used in another interval in the same multi-load pattern.

  When enabled, this switch has no effect on pattern simulation. However, the tool will transcript a warning message if a pattern violates the restriction.You can issue the set_clock_restriction command with only this argument, and the tool will retain whatever other clock restriction setting (Clock_po, On, Off or Domain_clock) is currently in effect.

  OFf — A literal that disables the requirement to use the same clock in every cycle within a load/unload interval during pattern creation. This is the invocation default.

  ON — A literal that specifies to use the same clock in every cycle within each load/unload interval during pattern creation.

Tip: If you want to create patterns that test only intra-clock domain faults, and you want to allow multiple non-interacting clock domains to be tested simultaneously in order to reduce pattern count, issue the following command prior to creating patterns:

```
set_clock_restriction domain_clock -any_interaction -same_clocks_between_loads on
```

- -IGNORE_unobservable_cells ON | OFf

  An optional switch and literal pair that bypasses checking any unobservable nonscan or scan cells for clock interaction purposes. You can issue the set_clock_restriction command with only this argument, and the tool will retain whatever other clock restriction setting (Clock_po, On, Off, or Domain_clock) is currently in effect.

  ON — A literal that enables bypassing the check for unobservable nonscan or scan cells.

  OFf — A literal that disables bypassing the check for unobservable nonscan or scan cells.

For example, if two clocks interact on a state element that is unobservable due to pin constraint or cell constraint, then the tool does not consider these two clocks incompatible for this statement when this switch is set to ON.

Example

The following example checks the current clock restriction setting, adds the additional restriction that the tool use the same clock in every cycle within each load/unload interval during pattern creation, then changes the basic setting from clock_po to “on”:

report_environment

```
...
clock restriction =       clock_po
...
```

set_clock_restriction -same_clock_between_loads on
report_environment

```
...
clock restriction =       clock_po, same_clocks_between_loads
...
```

set_clock_restriction on
report_environment

```
clock restriction =       ON, same_clocks_between_loads
```

Related Commands

add_norace_clocks

report_clock_domains

set_compactor_connections

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

Mode: setup

Usage

set_compactor_connections {-**CHANnel** ***channel_number***} {-**CHAIns** {***chain_name***…}}

Description

Specifies the scan chains for the tool to connect to a particular scan channel output.

The set_compactor_connections command specifies a list of scan chains whose outputs you want the tool to compact into the specified EDT scan channel output. The tool makes these connections using one or more stages of XOR gates in the compactor circuitry. If you use this command to specify the compactor connections for one scan channel and group of scan chains, you must use it for all the channels and chains in the design.

When generating the EDT IP, the tool uses the sequence of add_scan_chains commands to connect the EDT hardware to the scan chains. You can change the order of the add_scan_chains commands in your dofile to change how they are connected to the decompressor and compactor. However, this method changes both the decompressor and compactor connections for a particular chain.

Alternatively, you can use the set_compactor_connections command to override the tool's default connections and explicitly define the connections between scan chains and compactor. Using this command allows you to change the compactor connections without changing the default decompressor connections to those chains.

If you have dual configurations, you can still define the compactor connections using the set_compactor_connections command, but only for the configuration that uses all scan channels. The set_compactor_connections command is not tied to a specific configuration since you only need to define connections once for each channel. See Example 2.

Arguments

- **-CHANnel** ***channel_number***

  A required switch and positive integer pair that identify the scan channel to connect to the specified group of scan chains.

  When you specify the number of channels using the set_edt_options command -*Channels* switch, the tool numbers the channels consecutively starting at 1.

- **-CHAIns** ***chain_name***…

  A required switch and repeatable string pair that specify the internal scan chains whose outputs the tool should connect through the compactor circuitry to the specified scan channel output.

Example 1

For a design with eight scan chains named “chain1” through “chain8”, the following example sets up the EDT logic to have two scan channels. It then connects the outputs of chain1, chain5 and chain6 through the compactor to channel 1 and displays these connections. Any channel listed with “no connections” next to it still needs to be defined. The example goes on to define the remaining compactor connections and displays the completed configuration.

set_edt_options -channels 2
set_compactor_connections -channel 1 -chains chain1 chain5 chain6
report_compactor_connections

```
//  Reporting user-defined compactor connections:
//
//  Channel   Chains
//  -------   ------
//     1      chain1, chain5, chain6
//     2      no connections
//
```

set_compactor_connections -channel 2 -chains chain2 chain3 chain4 chain7 chain8
report_compactor_connections

```
//  Reporting user-defined compactor connections:
//
//  Channel   Chains
//  -------   ------
//     1      chain1, chain5, chain6
//     2      chain2, chain3, chain4, chain7, chain8
//
```

Example 2

In the following example, compactor connections are defined for a dual configuration. Note, the connections are only defined for the configuration that uses all scan channels.

```
set_current_edt_configuration config_high
set_edt_options -input 1 -output 1

set_current_edt_configuration config_low
set_edt_options -input 3 -output 3

set_compactor_connections -channel 1 -chains ...
set_compactor_connections -channel 2 -chains ...
set_compactor_connections -channel 3 -chains ...
```

Related Commands

report_compactor_connections
reset_compactor_connections
set_edt_options

set_config_value

Context: all contexts including unspecified

Mode: all modes

Usage

set_config_value ***config_spec*** [-in_wrapper *wrapper_object_or_name*]}
[-id {<*position*> | *id_name* | all} | -value_id {<*position*> | *value_name* | all}
{-specify_default | *value_or_value_list* | -unspecify} [-partition *partition*]

Description

Sets the value of an element in the configuration data tree.

Arguments

- ***config_spec***

  A required value that specifies the name of a configuration element or a collection containing one configuration element. When config_spec is a collection, the -in_wrapper cannot be used because the object contains all of the information about the configuration element. In order to use the -in_wrapper option, the config_spec value must be a name and the name must be relative to the specified wrapper element. You can use a positional id for any level of hierarchy. You can also replace the id of a leaf name by the position. For example, the path “tmp(1)/<0>/wrp<1>” means “tmp(1)/wrp(abc)/wrp(def)” if “wrp(abc)” is the first element found inside “tmp(1)” and “wrp(def)” is the second instance of a wrapper with a base name of “wrp” inside “tmp(1)/wrp(abc)”.

- -in_wrapper *wrapper_object_spec*

  An optional option value pair that is used to specify the parent wrapper in which the configuration element exist. The wrapper_object_spec value is the name of a wrapper or a collection containing a wrapper element. The names in the config_element_name is relative to the specified parent wrapper.

- -id *position* | *id_name* | all

  An optional option value pair that specifies that the data to set is the id of a wrapper or of a repeatable property. When using -id all, the command expects a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable Id declared for it. In this case the value is supplied as a tcl string which is the case for most wrappers and repeatable properties you will set.

  When you have configuration elements with more than one Id, you can set one id at a time using either its position or when it has metadata, the id of the Id wrapper that declares the id. A position is represented with an integer surrounded by <>. For example, <0> is the first id, <1> is the second, and so on.

- -value_id *position* | *value_name* | all

  A optional option value pair that specifies that the data to set is the value of a property, repeatable property or of a row inside a DataWrapper or a CsvWrapper. If the -id or -value_id options are not specified, “-value_id all” is assumed.

  When using “-value_id all”, the command expects a well-formatted Tcl list unless the configuration element has metadata and there is only one non-repeatable value declared for it. In this case, the value is expected to be a Tcl string which is the case for most properties you will set.

  When you have configuration elements with more than one value, you can set one value at a time using either its position or, when it has metadata, the id of the Value wrapper that declares the value. A position is represented with an integer surrounded by <>. For example, <0> is the first value, <1> is the second, and so on.

- -specify_default

  An optional Boolean option that specifies that the property is to be set to its default value. This option is only allowed on properties.

- value_or_value_list

  An optional string or Tcl list of strings defining the value at which to set the Ids or Values of properties or wrappers. You use a list only when using “-id all” or “-value_id all” unless the configuration element has metadata and there is only one non-repeatable id or value declared for it; otherwise, you use a string.

- -unspecify

  An optional Boolean option that specifies that the property is to be unspecified and thus return to its default value. This option is only allowed on properties.

- -partition_name *partition_name*

  An optional Boolean option that specifies that the data requested is the name of the partition in which the configuration element exists. Table 7-38 shows the returned value based on which partition the configuration element exists in.

  An optional option value pair that is used to specify the parent wrapper in which the configuration element exists. When the option is unspecified, the default partition is used which is where all specifications (DftSpecification and DefaultsSpecification/DftSpecification) are located. You can access the Core wrappers comprising the Tessent Core Description syntax in the tcd partition. Using the partition

names meta: and meta:tcd allows you to access the metadata configuration elements that define the default and tcd partitions. Table 7-38 summarizes the available partition names.

Table 7-38. Available Partition for Introspection

Partition name	Configuration data contained
	When the -partition option is unspecified, the Default partition is access which is where the DftSpecification and DefaultsSpecification/DftSpecification configuration data is located.
tcd	When the -partition option is specified with the tcd value, the Core wrappers comprising the Tessent Core Description syntax is accessed
meta	When the -partition option is specified with the meta value, the metadata configuration data defining the allowed syntax of the DftSpecification and DefaultsSpecification/DftSpecification is accessed.
meta:tcd	When the -partition option is specified with the meta:tcd value, the metadata configuration data defining the allowed syntax of the Core wrappers is accessed.

Example

The following example uses a foreach loop to set the properties inside a wrapper

```
> set int [get_config_el Interface \
 -in /DftSpecification(corea,rtl)/IjtagNetwork/HostScanInterface(ijtag)]
> foreach {prop val} {scan_in mySI scan_out mySO tck myTCK} {
  set_config_val $prop -in $int $val
}
> report_config_data $int

Interface {
  tck : myTCK;
  scan_in : mySI;
  scan_out : mySO;
}
```

Related Commands

get_config_value
get_defaults_value
report_config_data

set_contention_check

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

Mode: setup, analysis

Usage

set_contention_check {{**ON** | OFf | **CAPture_clock**} [-Warning | -Error] [-Bus | -Port | -ALl] [-BIdi_retain | -BIDI_Mask] [-NO_FLoating {BIdi_only | ALl_busses}] [-ATpg | -CATpg] [-NOVerbose | -Verbose | -VVerbose]}

Description

Specifies whether or not contention checking is on and the conditions under which checks are performed. Contention checking is set to On when the tool is invoked.

When a bused output of a tri-state driver (or switch) that is driving an X caused by an X on its enable is encountered, the contention is not reported if the data input to the driver is at the same level as other drivers on the bus. Also, the simulated value of the bus gate is resolved to a binary value if it can do so without tracing through additional bus gates.

Arguments

- **ON**

  A literal that performs contention checking during simulation. The tool does not propagate captured data effects. This is the default.

  If the set_split_capture_cycle command is on and there are multiple capture clocks in the design, the set_contention_check ON command performs a contention check not only before the capture but after the capture as well.

- **OFf**

  A literal that specifies to not perform contention checking during simulation.

> **Tip**: If you create patterns using ATPG Expert (“create_patterns”), it is automatically determined whether or not the -Atpg switch is necessary and is used only when needed; therefore, you do not need to specify “set_contention_check on -atpg”.

- **Capture_clock**

  A literal that performs contention checking both with and without propagating captured data effects. If a clock, read control, or write control line connects to a bus, the tool also performs bus contention checking with all clocks off prior to the application of the capture clock. The tool does not consider any contention patterns for fault simulation and does not place any of these patterns into the internal test pattern set.

Tip: If you create patterns using ATPG Expert ("create_patterns"), it will automatically determine if the -Catpg switch is necessary and use it only when it is needed; therefore all you need to specify is "set_contention_check capture_clock".

- -Warning

  An optional switch that displays a warning message, but continues simulation if bus contention occurs during simulation. The warning message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- -Error

  An optional switch that displays an error message and stops the simulation if bus contention occurs.

  You can debug contention errors by using the -Error switch to stop simulation at the point of the first contention error.

  Using this option, you can then view the simulated values of all gates in the first bus contention pattern by using the report_gates command.

  The error message indicates the number of patterns the tool rejected in the current simulation pass of 64 patterns and also identifies the bus gate on which the bus contention occurred.

- -Bus

  An optional switch that performs contention checking of tri-state driver buses. This is the default.

  Tri-state logic allows several bus drivers to time-share a bus. However, if the circuit enables two bus drivers of opposite logic to drive the bus, physical damage can occur. This switch allows the tool to identify these conditions and notify you of their existence.

  The tool identifies buses which have circuitry that prevents bus contention and does not check for bus contention problems. This eliminates false bus contention reporting when multiple inputs to a bus are at X. Bus contention that occurs on weak buses does not result in an E4 rules checking violation or pattern rejection during simulation. The tool continues to simulate them as an X state.

- -Port

  An optional switch that performs contention checking for multiple-port flip-flops and latches. The tool identifies and rejects patterns during which any multiple-port latch or flip-flop has more than one clock, set, or reset input active (or at X).

- -BIDI_Retain

  An optional switch that rejects patterns during contention checking that cause the direction of bidirectional pins to change following the capture clock.

- -BIDI_Mask

  An optional switch that causes the fault simulator to modify the input values of bidi pins to avoid bus contention after the capture clock if a device pin changes from input mode to output mode as the clock is applied.

- -NO_FLoating {BIdi_only | ALl_busses}

  An optional switch and literal pair that checks for floating buses in addition to contention when bus contention checking is in effect. The Bidi_only option only checks for floating at buses associated with bidirectional pins. The All_busses option checks for floating at all buses. This switch enables the E11 DRC with handling set to warning if it was previously set to ignore. (If you have previously set E11 handling to note or error, your setting is retained).

  The floating bus checks are always performed during pattern simulation when bus contention checking is in effect. If -Atpg or -Catpg is included, floating buses are avoided during test generation.

  The following rules apply:

  - Floating checks are performed for strong buses only, not weak buses.
  - If a strong bus passes the E11 DRC, the switch does not apply to that bus.
  - If a strong bus drives a pull bus, the strong bus is allowed to float if the weak input of the pull bus is set to a known binary value.
  - This switch and the -Bidi_mask switch cannot be used together in the same command.
  - This switch does not require that a bus have a known binary value to pass the floating check. The bus may be driven to an unknown binary value.
  - If a bus is driven by a TIEX gate directly, the TIEX gate is treated as a source that can take on a Z value. As a result, it is impossible to avoid floating this bus and ATPG cannot generate any test patterns.

- -ALl

  An optional switch that performs contention checking for both tri-state driver buses and multiple-port flip-flops and latches.

- -ATpg

  An optional switch that forces all buses to a non-contention state, which ensures that test patterns are not created causing bus contention.

  After completing normal test pattern generation for a fault, the tool forces all buses to a non-contention state. If the tool cannot satisfy this condition, given the conditions set by the original pattern, the tool aborts the fault, excludes the pattern from the final test set, and displays a message indicating the number of these aborted faults for each simulation pass. No attempt is made to change the original pattern.

The -Atpg option results in additional effort by the test pattern generator and you should use it only when necessary.

Tip: If you create patterns using ATPG Expert ("create_patterns"), you may get the best results by not explicitly specifying the -Atpg switch. ATPG Expert will automatically determine if the switch is necessary and utilize it only when it is needed.

- -CATpg

  An optional switch that uses clock sequential ATPG effort in the capture cycle as well as all other cycles. If neither this switch nor the -Atpg switch is specified, the default will continue to be that there is no ATPG effort for bus contention in any cycle, and ATPG constraints are satisfied deterministically prior to the capture clock event only. Specifying the -Catpg switch without the Capture_clock option is equivalent to using just the -Atpg switch.

Tip: If you create patterns using ATPG Expert ("create_patterns"), you may get the best results by not explicitly specifying the -Catpg switch. ATPG Expert will automatically determine if the switch is necessary and utilize it only when it is needed.

- -Verbose

  An optional switch that reports the first reason for each pattern rejected (maximum of 64 messages per parallel pattern invocation).

- -VVerbose

  An optional switch that reports every reason for each pattern rejected (there is no limit to the number of messages per parallel pattern invocation). The messages will be of the same type issued for the -Verbose switch.

Note

For large designs, this option may produce thousands of lines of output for each pattern simulated.

- -NOVerbose

  An optional switch that lets you turn off the -Verbose or -VVerbose effects that may have been set previously.

Example 1

The following example performs contention checking on multiple-port flip-flops and latches, stops the simulation if any bus contention occurs, and displays an error message. The message indicates the number of patterns rejected and the bus gate on which the bus contention occurred:

```
set_contention_check on -port -error
set_system_mode analysis
create_patterns
```

Example 2

The following example performs contention checking on both multiple-port sequential gates and tri-state buses, stops the simulation if any bus contention occurs, and displays an error message that will indicate the gate on which the contention occurred:

```
set_system_mode analysis
set_contention_check on -all
create_patterns
```

Related Commands

report_gates
set_gate_report
set_bus_handling

set_context

Context: unspecified, all contexts

Mode: all modes

Usage

set_context **dft** [-scan | -edt | -logic_bist | -test_points] [-rtl | -no_rtl] [-license *license_name*] [-design_identifier *string*]

set_context **patterns [**-ijtag | -scan | -scan_retargeting | -scan_diagnosis | -failure_mapping**]** { [-rtl] | [-no_rtl] } [-license *license_name*] [-design_identifier *string*]

Description

Specifies the current usage context of Tessent Shell.

You must set the context before you can invoke most other commands in Tessent Shell.

Arguments

- **dft**

  A required literal that sets the context to dft. In the basic dft context, you load your design in setup mode and then move to the insertion mode to execute design editing commands such as create_connections, create_instance, and move_connections. When used with the -scan or
  -edt options, the dft context is used to insert scan chains or insert the EDT controller.

- **patterns**

  A required literal that sets the context to patterns which provides functionality related to scan and IJTAG pattern generation, ICL extraction, and scan pattern diagnosis.

- -scan

  An optional switch that is available in both patterns and dft contexts. This sub-context specifies ATPG and simulation functionality for standard ATPG, TestKompress, and LogicBIST patterns when specified with the patterns context, and for scan insertion when specified with the dft context.

- -edt

  An optional switch that is available in dft context. This sub-context specifies to create and insert EDT IP into the design. When you provide a scan-inserted netlist, the tool analyzes the scan chains and generates and inserts an appropriate EDT controller.

- -logic_bist

  An optional switch that is available in the dft context and must be specified in conjunction with the -edt switch. This switch enables the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

- -test_points

  An optional switch that is available in the dft context and specifies functionality for test point identification and insertion.

- -ijtag

  An optional switch that is available only in patterns context. This sub-context specifies functionality related to the IEEE P1687 standard such as generating IJTAG patterns by executing PDL commands such that they are retargeted to the current design boundary, and performing ICL extraction. When the set_current_design command is executed, the presence of an ICL module matching the specified current design name sets the context to ijtag retargeting. If there is no ICL module matching the specified current design name, the ICL extraction mode is enabled. Executing get_context -extraction returns 1 when in the extraction mode.

- -scan_retargeting

  An optional switch that is available in the patterns context. This sub-context enables scan pattern retargeting functionality for retargeting core-level test patterns at the top level. For more information, see "Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

- -failure_mapping

  An optional switch that is available only in the patterns context. This sub-context specifies to reverse map top-level failure files for retargeted scan pattern results, which were generated from within the -scan_retargeting sub-context. Reverse mapping results in a set of core-level failure files that you can diagnose with Tessent Diagnosis. For more information, refer to the *Tessent Diagnosis User's Manual*.

- -rtl | -no_rtl

  In dft context, these switches specify whether you are performing netlist editing or RTL editing. You must use the -rtl option if you are editing RTL. You can use either option when editing a netlist but be aware that the write_design usage model is different between these modes. If you want to write out your edited netlist into a single file, select the -no_rtl option.

  In the patterns -ijtag context, the -rtl switch is inferred if coming from the dft context and the -rtl option was used when setting the dft context. When entering the patterns –ijtag context from the unspecified context, -no_rtl is assumed. ICL Extraction makes use of quick synthesis to convert any RTL in the fanin or fanout of ICL modules. See the description of the synthesize_before_analysis Module attribute if you are using a test_setup procedure for ICL extraction.

  In all other contexts, the -rtl switch is not allowed and the -no_rtl switch is optional because it is always inferred.

  In patterns -ijtag context with RTL support, during ICL extraction the input cones in front of design ports with matched ICL attributes are synthesized and flattened in order to perform extraction of the paths and end points, which results in an extracted ICL top level description. Checking and debugging of trace issues, ijtag related checks and ICL semantic

checks are also done at this stage. The same is true for output cones driven by design ports with matched ICL attributes.

- -license *license_name*

  An optional switch and string pair that specifies a particular license to use. By default, Tessent Shell acquires a license from the available product that provides the requested functionality/context. Table 7-39 lists the supported license names for the available contexts and sub-contexts.

Table 7-39. Contexts Available by License

Context/Sub-Context	License Name(s)
dft -rtl dft -no_rtl	DesignEditor, BoundaryScan, SocScan, IJTAG, PLLTest, MemoryBIST, SerdesTest, TestKompress, LogicBIST
dft -edt	TestKompress
dft -logic_bist -edt	LogicBIST
dft -scan	Scan, SoCScan
dft -test_points	SoCScan, LogicBIST
patterns -ijtag	IJTAG BoundaryScan, SocScan, PLLTest, MemoryBIST, SerdesTest, TestKompress, LogicBIST
patterns -scan	When EDT is used: TestKompress When EDT and LBIST are not used: TestKompress, FastScan When LBIST is used: LogicBIST
patterns -scan_retargeting	TestKompress
patterns -failure_mapping	Diagnosis
patterns -scan_diagnosis	Diagnosis

  If Tessent Shell is unable to acquire a license when setting the context, the tool returns an error message.

- -design_identifier *string*

  An option used to assign an identifier to the current design. If this option is unspecified, the identifier defaults to “rtl” in the -rtl context, and to “gate” in the -no_rtl context.

  You only need to specify this option in the dft context when you are doing a second insertion pass on the same design. When you use the process_dft_specification command to insert all instruments in one insertion pass, the default value is adequate. In the dft context, the tool generates an error if module names matching <current_design>_<design_id>_tessent_* are found already instantiated into the design during transition to analysis mode or when running the process_dft_specification command; this indicates that you are doing a second insertion pass into that same design and that a new design identifier value needs to be specified.

Example 1

The following example demonstrates setting the context to dft. In this case, one of the dft licenses is checked out if one is not already checked out.

```
set_context dft -no_rtl
```

Example 2

The following example demonstrates setting the context to patterns -scan and requesting a TestKompress license. If the -license option is not specified, a FastScan license is first checked out if one is available. After that, a check is performed to determine whether LBIST or EDT modes are used; if they are, the FastScan feature is released and either the TestKompress or LBIST license is checked out, as appropriate.

```
set_context patterns -scan -license TestKompress
```

Related Commands

report_context

get_context

set_current_design

Context: all contexts

Mode: setup, insertion

Usage

set_current_design [*top_design*]

Description

Specifies the top level of the design for all subsequent commands until reset by another execution of this command.

When you issue this command without an argument, the tool automatically finds the top level of the design if only one exists. If the tool finds more than one top, the tool presents a list of the possible top levels; you must reissue the command and specify one of the top levels listed.

Note that parameterized views of modules are only available to introspection if the parameterized module is instantiated below the current design. You can only set the current design to a module with default parameters, or to a non-parameterized module.

If you want to read additional designs or libraries after setting a top module, you must again issue the set_current_design command to re-elaborate the design before you can operate on the design.

If you have operated on the design or have read a flat model, you cannot read additional designs or libraries, or set a new top design without either deleting the current settings using the delete_design command or using the “read_verilog -force” or “read_cell_library -force” command.

In the dft and “patterns -ijtag” context, the set_current_design command is used to set the current design even if the design only has an ICL view. When set_current_design executes in these contexts, if a loaded ICL module exists whose name matches the specified current design name, the tool sets the context to ijtag retargeting; if no loaded ICL module exists whose name matches the specified current design name, the tool enables the ICL extraction mode.

In the ijtag retargeting context, if a design module is not already loaded, the tool creates a design module with the ports found in the ICL module. If a design module is already loaded, the tool verifies that the module contains all of the ports defined in the ICL module.

The tool writes out patterns that include the list of ports found within the design module as well as those in the ICL module. If you plan to use a test_setup procedure that interacts with ports that are in the design module but not described in the ICL module, you must load the design view and not allow it to be created from the ICL module.

Arguments

- *top_design*

  An optional string that specifies the name of the top-level module in the design. When the module name is provided, the set_current_design command looks for the module in the

default library. If you want to make a module from a different library the current design, you can use the get_modules command to return the module and then pass the module object directly to set_current_design as follows:

set_current_design [get_modules *module_name* -in_library *library_name*]

Example 1

The following example sets the top design to block_1:

set_current_design block_1

```
//  Note: Top design is 'block_1'.
```

Example 2

The following example finds the top level of the design (block_2) and sets it as the top. This use model only works in the no_rtl context and requires that all modules currently loaded are instantiated below a single module.

set_current_design

```
//  Note: Top design is 'block_2'.
```

Example 3

The following example loads a mixed-language design in the DFT context in RTL mode.

SETUP> **set_context dft -rtl**

```
# Specify library1 and work as the two active libraries
```
SETUP> **set_logical_design_libraries -logical_names {library1 work} -default_library work**

SETUP> **read_vhdl core*.vhdl -in_library library1 -format 93**

```
# Set the current_design to core1 from library1
```
SETUP> **set_current_design [get_modules core1 -in_library library1]**

```
//  Note: Top design is 'core1'.
```

Related Commands

delete_cell_library
delete_design
read_flat_model
read_cell_library
read_verilog
set_cell_library_options
set_design_include_directories
set_design_macros

set_current_edt_block

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

Mode: setup, analysis

Usage

set_current_edt_block ***block_name***

Description

Directs the tool to apply certain subsequent commands to a particular EDT block only.

The set_current_edt_block command restricts the applicability of certain subsequent commands to the EDT block having the specified *block_name*. This is referred to as setting or changing the EDT context and the EDT block on which context-sensitive commands currently operate is referred to as the current EDT block. The current EDT block remains the sole target of the context-sensitive commands until you delete its definition using the delete_edt_blocks command, or you change the context to a different EDT block using the add_edt_blocks command or another set_current_edt_block command.

Use this command when you want to define or redefine the attributes of a specific EDT block using context-sensitive commands. The add_scan_chains command, EDT-specific commands that define attributes for a given block, and some reporting commands are presently the only context-sensitive commands. All other commands (set_system_mode, create_patterns and report_statistics for example) apply to the entire netlist.

If multiple EDT blocks are defined and none of them is specified as the current EDT block, subsequent context-sensitive commands that define attributes for a given block ("set_edt_options -channels" for example) will return an error. This can happen if you add multiple EDT blocks, then delete the one that is the current EDT block.

For more information about EDT blocks or the modular Tessent TestKompress flow, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Arguments

- ***block_name***

  A required string that specifies the name of the EDT block you wish to set as the current EDT block for subsequent context-sensitive commands. The *block_name* must be one you defined previously using the add_edt_blocks command. To view a list of the current defined block names, enter "report_edt_blocks".

Example 1

The following example lists current user-defined block names, then changes the EDT context to the EDT block named "my_core1_block":

report_edt_blocks

```
//  my_core1_block
```

```
//  my_core2_block
//  my_core3_block
//  my_core4_block   (current block)
```

set_current_edt_block my_core1_block
report_edt_blocks

```
//  my_core1_block   (current block)
//  my_core2_block
//  my_core3_block
//  my_core4_block
```

Example 2

The following example uses the set_current_edt_block command to force the context-sensitive report_edt_configuration command to report on specific EDT blocks:

set_current_edt_block my_core2_block
report_edt_configuration

```
//
//  EDT block "my_core2_block"
//  -------------------------
//  IP version:                     3
//  External scan channels:         3
//  Decompressor size:              10
//  Clocking:                       edge-sensitive
//
//  Note: Only current EDT block "my_core2_block" is reported. Use "report
//  edt configuration -all_blocks" to report the configurations of all
//  blocks.
```

set_current_edt_block my_core4_block
report_edt_configuration

```
//
//  EDT block "my_core4_block"
//  -------------------------
//  IP version:                     3
//  External scan channels:         1
//  Decompressor size:              8
//  Clocking:                       edge-sensitive
//
//  Note: Only current EDT block "my_core4_block" is reported. Use "report
//  edt configuration -all_blocks" to report the configurations of all
//  blocks.
```

Related Commands

add_edt_blocks
delete_edt_blocks
report_edt_blocks

set_current_edt_configuration

Context: dft -edt

Mode: setup

Prerequisites: One or more compression configurations must be defined with the add_edt_configurations command.

Usage

set_current_edt_configuration ***config_name***

Description

Sets a particular compression configuration for the tool to use within the current design block.

This is necessary so that subsequent commands can be applied to define or redefine the attributes of the specific compression configuration.

For more information about compression configurations, see the "Defining Dual Compression Configurations" section of the *Tessent TestKompress User's Manual*.

Arguments

- ***config_name***

  A required string that specifies the name of a compression configuration. The *config_name* is the name specified when the compression configurations were defined. Use the report_edt_configurations command to display a list of the currently defined configuration names.

Example

The following example sets the current EDT block, *my_core1_block*, and the compression configuration, *system_test.* Then, it redefines the input and output channels and reports the new configurations.

set_current_edt_block my_core1_block
report_edt_configuration

```
// my_core1_block
// IP version: 4
// Bypass logic: On
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10] [system_test : 4]
// Output channels:[manufacturing_test : 5] [system_test : 2]
// Bypass chains: 5
```

set_current_edt_configuration system_test
set_edt_options -input_channels 3 -output_channels 3
report_edt_configuration

```
// my_core1_block
// IP version: 4
// Bypass logic: On
```

```
// Lockup cells: On
// Clocking: edge-sensitive
// Input channels: [manufacturing_test : 10][system_test : 3]
// Output channels: [manufacturing_test : 5][system_test : 3]
// Bypass chains: 5
```

Related Commands

add_edt_configurations
delete_edt_configurations
report_edt_configurations

set_current_mode

Context: patterns -scan, patterns -scan_diagnosis, patterns -scan_retargeting, patterns -failure_mapping

Mode: setup

Usage

For -scan and -scan_diagnosis

set_current_mode [mode_name] [-type mode_type]

For -scan_retargeting and -failure_mapping

set_current_mode mode_name

Description

Specifies the current mode.

Arguments

- mode_name

  An optional or required user-defined literal that specifies the name of the mode. If not specified, the mode_name argument defaults to the value of mode_type. The mode_name is optional for the -scan and -scan_diagnosis contexts and required for the -scan_retargeting and -failure_mapping contexts.

- -type mode_type

  An optional switch that specifies the mode type when generating retargetable patterns. Valid options are:

  unwrapped — Specifies the current default test mode that ATPG runs in with Tessent FastScan and Tessent TestKompress. This mode does not support core isolation or hierarchical test.

  internal — Specifies the internal test mode of a core in which patterns are generated while the core is isolated from its surroundings. Due to this isolation, those patterns are independent of the core's embedding in the design and can be retargeted from the core level to a higher level of the design. When this option is specified, the tool automatically adds constraints to primary inputs and masks to primary outputs during design rule checking. For more information, see "Core-Level Pattern Generation" in the *Tessent Scan and ATPG User's Manual*.

  external — Specifies the external test mode in which the periphery of the wrapped core is tested with the interconnect logic, in the context of its parent. This mode will eventually support the hierarchical integration of scan chains.

Example

This example sets the mode name and mode type to "internal".

```
set_current_mode -type internal
```

Related Commands

get_current_mode

set_current_simulation_context

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

set_current_simulation_context ***context_name***

Description

Sets the current simulation context.

The set_current_simulation_context command sets the simulation context for later analysis and introspection. Table 7-40 lists the predefined simulation contexts.

Table 7-40. Predefined Simulation Contexts

Simulation Context	Description
stable_after_setup	Specifies the values that are constant in the period following the end of the test_setup procedure and the beginning of the test_end procedure. For scan patterns, the value is constant throughout scan and capture cycles. For IJTAG patterns, the value is constant throughout any IJTAG operations.
stable_capture	Specifies the values that are constant throughout the capture procedures.
stable_load_unload	Specifies the values that are constant throughout the load_unload cycles.
stable_shift	Specifies the values that are constant throughout the shift cycles.

Arguments

- ***context_name***

  A required literal that specifies the simulation context to make current. You can specify any of the simulation contexts listed in Table 7-40 or any user-defined simulation contexts that have been added. By default, the simulation context is stable_after_setup.

Example

The following example sets the current simulation context to stable_shift:

```
> set_current_simulation_context stable_shift
```

Related Commands

get_current_simulation_context
get_simulation_context_list
report_simulation_contexts

set_decision_order

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

Mode: setup, analysis

Usage

set_decision_order **-NORandom** | **-Random**

Description

Specifies how the ATPG selects gate input or output nodes for evaluation during ATPG.

The set_decision_order command specifies whether ATPG chooses a gate input or output for evaluation randomly, or by its ease of evaluation. Ease of evaluation for a gate input depends on how easy it is for the tool to control the input. For a gate output, ease of evaluation is related to how easily the tool can observe the output. A nonrandom decision order causes the tool to evaluate the “easiest” gates first.

Arguments

- **-NORandom**

  A required switch that causes the ATPG to choose the easiest evaluation when faced with a decision about which gate input or output to evaluate next. This is the default unless you are using the create_patterns command, which always uses a random decision order.

- **-Random**

  A required switch that causes ATPG, when faced with a decision about which gate input or output to evaluate next, to make random decisions rather than choosing the easiest evaluation. This causes variation between patterns, tending to give a more compact vector set at the risk of creating more ATPG aborts.

Note

In some cases, depending on circuit topology, changing the order in which the tool makes decisions during ATPG may improve performance. If large numbers of faults are being aborted, for example, you may eliminate or reduce aborted faults by setting the decision order to “random” without changing the abort limit.

Related Commands

set_atpg_compression

create_patterns

set_defaults_value

Context: dft

Mode: analysis

Usage

set_defaults_value **property_path_name** {*value* | -unspecify} [-policy *policy*]

Description

Sets a property inside one of the DefaultsSpecification/DftSpecification (*policy*) wrappers.

The property_path_name is the pathname of the configuration without the DefaultsSpecification/DftSpecification(*policy*) prefix.

When the -policy option is not specified, the tool populates the user policy. If you use the -unspecify option, the specified property_path_name property is set to unspecified in the appropriate DefaultsSpecification/DftSpecification(cation*policy*) wrapper.

Arguments

- **property_path_name**

  A required string that specifies a configuration pathname without the DefaultsSpecification/DftSpecification(*policy*) prefix of the property you want to set in one of the DefaultsSpecification/DftSpecification(*policy*) wrappers.

- *value*

  An optional value that specifies the value of the property you want to set.

- -unspecify

  An optional value that specifies to unspecify the property instead of setting its value.

- -policy *policy*

  An optional switch value pair that specifies on which DefaultsSpecification/DftSpecification(*policy*) wrapper the property is to be set or unspecified. When this switch is not specified, the user policy is used.

Example

The following example sets the rtl_extension property in the company Dftspecification policy and unspecifies the same property in the user policy.

```
> set_defaults_value Dftspecification/rtl_extension vb -policy company
> set_defaults_value Dftspecification/rtl_extension -unspecified  -policy user
> get_defaults_value Dftspecification/rtl_extension
vb
```

Related Commands

get_defaults_value

create_dft_specification

set_design_include_directories

Context: unspecified, all contexts

Mode: setup

Usage

set_design_include_directories *include_dir* … | [-Clear]

Description

Specifies one or more directory paths that contain `include files for use by subsequent read_verilog commands.

This command provides equivalent functionality to the –Incdir invocation switch. If you issue this command multiple times, the last set of include directories is used. When you issue this command without arguments, the command reports the list of include directories.

Each include_directory must be relative to the current tool invocation directory or an absolute directory pathname. Include files are searched for in the following order of precedence:

1. Absolute pathnames specified by `include directives in the Verilog design.
2. Directories specified with the command.

   For relative `include file targets, directories are searched in the order they are listed, left to right, in the command. If identically named files exist in multiple directories, the first file found is used and the others are ignored.
3. Directory where the Verilog design file is located.
4. Directory in which you invoked the tool.

Arguments

- *include_dir*

  A string that specifies one or more directory paths.
- -Clear

  An optional switch that removes the current list of include directories.

Example

The following commands specify that the tool search for `include files contained in ../dir1 when loading the top.v module:

```
set_design_include_directories ../dir1
read_verilog top.v
```

Related Commands

delete_design
read_verilog
set_current_design
set_design_macros
read_cell_library
read_flat_model

set_design_level

Context: dft, patterns

Mode: setup, insertion

Prerequisites: You can set the level can only after the design is elaborated.

Usage

set_design_level {**chip** | **physical_block** | **sub_block**}

Description

Specifies the level of the current design.

You must specify the design level with this command when you are in the “dft -rtl” or “dft -no_rtl” contexts and want to change to analysis system mode. There is no default setting, so you must specify the appropriate level because this setting affects the create_dft_specification and process_dft_specification commands, as well as some design rules checks that the tool performs when changing to the analysis system mode.

If you issue the extract_icl command immediately after having issued the process_dft_specification command, and if you specified the design level as chip or physical_block while in dft context, the tool uses the specified design level during extract_icl to set the tessent_is_physical_block attribute to true on the top ICL module. If you did not run the extract_icl command in the same tool session as the dft step, you must set the design level prior to issuing the extract_icl command.

Arguments

- **chip** | **physical_block** | **sub_block**

  A required literal that specifies the design level:

  chip — The design is intended to be used at the chip level. This implies that all ports have a pad buffer and a TAP is inserted during the execution of the create_dft_specification command. A chip is by definition also a physical block.

  physical_block — The design is intended to be used for blocks that are synthesized and laid out as independent blocks.

  sub_block — The design is intended to be used for blocks that are synthesized and laid out as part of a parent physical block.

  Note that there is no default setting, so you must specify one of these design levels.

Example

The following example sets the level of the current design to physical_block.

```
set_design_level physical_block
```

Related Commands

get_design_level

set_current_design

set_design_macros

Context: unspecified, all contexts

Mode: setup

Usage

set_design_macros *macro_name*[=*macro_value*] … | [-Clear]

Description

Specifies one or more defined macros for use by subsequent commands.

This gives you the convenience of using macros rather than having to edit the netlist to explicitly add the `define statements to control which parts of `ifdef statements are compiled. This command provides functionality equivalent to the –Define invocation switch.

If you issue the command multiple times, then the last set of macros is used. When you issue the command without arguments, the command reports the list of defined macros, including the default macros MGC_TESSENT and SYNTHESIS.

Arguments

- *macro_name*[=*macro_value*]

  An optional string that specifies one or more macros.

- -Clear

  An optional switch that clears the current list of all defined macros.

Example

The following commands specify that the tool use a custom macro named macro_1 set to a value of 20 when loading the top.v module:

```
set_design_macros macro_1=20
read_verilog top.v
```

Related Commands

delete_design
read_verilog
set_current_design
read_cell_library
read_flat_model

set_design_sources

Context: unspecified, all contexts

Mode: setup

Usage

set_design_sources [{{-format verilog} | {-format icl [-search_design_load_path On | Off]}}] [{-v *file_path* ...} {{-y *dir_path*... }... {-extensions *ext*...} }] [-clear]

Description

Specifies where the tool should look for the definition of undefined modules in the list of files specified by the read_verilog and read_icl commands.

When you issue the command without arguments, the command reports the list of –v/-y files and directories in their search order.

The -v, -y and -extensions switches have similar behavior to standard Verilog compilers like ModelSim.

When you issue the command multiple times, the tool uses the last set of –v/-y files and directories. You can overwrite –v files only when you issue the command with the –v switch. Similarly, you can overwrite –y directories only when you issue the command with the –y switch. Issuing the command with both –v and –y switches overwrites the previous files/directories.

Arguments

- -format verilog

  An optional switch that applies the specified search path to Verilog modules. This is the default. This switch cannot be used with the -search_design_load_path argument.

- -format icl

  An optional switch that specifies that the specified search path applies to ICL modules. This switch can be used with the -search_design_load_path argument.

  The ICL search path is used by the read_icl command to locate undefined modules found within ICL files that are read in; it is also used to locate ICL modules corresponding to design module names in ICL extraction mode,

- -search_design_load_path On | Off

  An optional switch and literal pair that specifies whether the directory into which design modules are loaded or searched for are automatically used as valid search directories for the ICL file. This option is On by default.

- -v *file_path*

  An optional switch that specifies one or more files in which to search for definitions of undefined modules. You can specify multiple –v, multiple –y, or both. The order the –v/-y files and directories specified in the command determines the search order during parsing.

- -y *dir_path*

  An optional switch that specifies one or more directories in which to search for definitions of undefined modules. Any file inside the –y directory should be named as module_name.extension in order to be parsed. You can specify multiple –v, multiple –y, or both. The order the –v/-y files and directories specified in the command determines the search order during parsing.

- -extensions *ext*

  An optional switch and value pair that instructs the tool to check only the files with the specified extensions. The value is a single extension name or a Tcl list of extension names. The tool supports compressed files by expanding the ext name or list of extension names to include those names with a .gz suffix. That is, if you specify "-extensions v", the tool expands it to "-extensions v v.gz vhd". You can use this switch only once per command execution, and the switch is required when you use the -y switch.

- -clear

  An optional switch that clears the list of files and directories previously specified with the -v or -y switches. The -clear switch operates on only one format per command execution. If you do not use the -format switch, the -clear switch operates on Verilog files.

Example 1

To illustrate how the switches work, consider the following example:

top.v	**f1.v**	**directory d1**
module top;	module foo();	foo.v bar.v foo.sv zar.v
foo i();	endmodule	
endmodule		
	module bar();	
	endmodule	

When you issue these commands:

```
set_design_sources -v f1.v
read_verilog top.v
```

then the module foo in file f1.v is parsed, but not module bar in f1.v.

When you issue these commands:

```
set_design_sources -y d1 –extensions v
read_verilog top.v
```

then the file foo.v is parsed, but not the other files in the d1 directory. Note that file foo.v is parsed but not foo.sv because the specified -extensions is v. In the previous example, when you issue the following commands:

```
set_design_sources -y d1 –extensions v –v f1.v
read_verilog top.v
```

then foo.v in d1 directory is parsed. If the command is

set_design_sources -v f1.v -y d1 –extensions v

then module foo in f1.v is parsed.

Example 2

In the following example, the set_design_sources command specifies to look for undefined module definitions in the dir1, dir2, and dir3 directories and to check only those files with a v_et or v extension.

set_design_sources -y {dir1 dir2 dir3} -extensions {v_et v}

Related Commands

delete_design
get_design_sources
read_verilog
report_design_sources
set_current_design
set_design_include_directories
read_cell_library
read_icl

set_dft_enable_options

Context: patterns -scan

Mode: setup

Usage

set_dft_enable_options **-type** {**xbounding** | **control_point** | **observe_point** | **mcp_bounding**}
-pin_name ***pin_name***
-value {**on** | **off**}

Description

Enables or disables the control or observe points.

Note that disabling either X-bounding or MCPs for LogicBIST is not allowed since it can result in Xs being captured in the scan cells.

Disabling the control or observe points enable signal can have an impact on coverage.

Arguments

- **-type xbounding** | **control_point** | **observe_point** | **mcp_bounding**

  A required switch and literal pair that specifies the type of point to either disable or enable.

- **-pin_name** ***pin_name***

  A required switch and string pair that specifies the name of the pin to enable or disable.

- **-value** {**on** | **off**}

  A required switch and literal pair that enables (on) or disables (off) the action for the specified pin.

Example

This example turns off observe points for transition fault simulation.

```
set_dft_enable_options -type control_point\
        -pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/control_point_en -value on

set_dft_enable_options -type observe_point \
      -pin_name /my_core_lbist_i/my_core_lbist_ctrl_signals_i/observe_point_en -value off

// Warning: Disabling 'observe_point' enable signal may have an impact
// on coverage.
```

Related Commands

set_lbist_controller_options

set_lbist_pins

set_diagnosis_options

Context: patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_diagnosis_options
[-Mode **Auto** | **Scan** | **Chain** | **Iddq**]
[-Verify_patterns **ON** **[Noverbose** | **Verbose]** | **OFf**]
[-Report **Default** | ***percentage***]
[-AT_speed **ON** | **OFf**]
[-MAx_suspects **Default** | ***limit***]
[-Time_limit **OFf** | ***seconds***]
[-WALL_time_limit **OFf** | ***seconds***]
[-Pattern_sampling {***failing_pattern_limit*** | **OFf**} {***passing_pattern_limit*** | **OFf**}]
[-FAILurefile_mismatch_verbosity **Default** | ***number*** | **All**]
[-BRidge_analysis **ON** | **OFf]**
[-CHEck_last_shift_value **ON** | **OFf**]
[-B2X_mismatch **Warning** **[Noverbose** | **Verbose]** | **Error**]
[-X2B_mismatch **Warning** **[Noverbose** | **Verbose]** | **Error**]
[-CELl_internal_analysis **ON** | **OFf**]
[-MAX_Faulty_chains *failed_chain_threshold*]
[-Chain_diagnosis_result **CEll** | **CHain**]
[-CHAIN_LIB_internal_pathname **OFf** | **ON**]
[-ABORT_DIAGNOSE_COMPOUND_faults **ON** | **OFf**]
[-ABORT_DIAGNOSE_MANY_faulty_chains **ON** | **OFf**]
[-ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability ***number***]
[-ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability ***number***]
[-IGNORE_TOOL_version **OFf** | **ON**}
[-INCLude_fail_signatures_size ***maximum_rows_per_table*** | **MAX**]
[-INCLUDE_BRIDGE_to_power **ON** | **OFf**]
[-GRoss_delay **ON** | **OFf**]
[-COMPOUND_Hold_time_fault_diagnosis **ON** | **OFf**]
[-INCLUDE_DFM_rules **ON** | **OFf**]
[-INCLUDE_RCD_constants **ON** | **OFf**]
[-EXPected_value **ON** | **OFf**]
[-internal_failing_cells **ON** | **OFf**]
[-cycle_offset *N*]

Description

Specifies the type of diagnosis performed by the diagnose_failures command.

The set_diagnosis_options command specifies global settings that determine the behavior of subsequent diagnose_failures commands. These global settings remain in effect until another set_diagnosis_options command is issued.

The set_diagnosis_options command is optional, as there are invocation defaults for all the settings. If no arguments are specified, a list of the current settings displays.

Arguments

- -Mode **Auto** | **Scan** | **Chain** | **Iddq**

  A switch and literal pair that determines what type of diagnosis is performed. The following options are available:

  Auto — A literal that enables Tessent Diagnosis to determine what type of diagnosis to run based on the contents of the failure file. If the failure file contains a *chain_test* keyword and chain test failures, a chain diagnosis is performed. If the failure file contains scan test failures, a scan diagnosis is performed.

  If the failures include a padding cycle of a scan chain, a chain diagnosis is run.

  If chain failures are not preceded by a *chain_test* keyword, use the -faulty_chain switch with the diagnose_failures command to run chain diagnosis.

  If the failure file contains both chain failures and scan failures, you must use the scan_test and chain_test keywords to identify the data. This is the default.

  Scan — A literal that enables scan diagnosis only. If the failure file contains scan test failures, a scan diagnosis is performed.

  If a padding cycle of a scan chain fails, the scan chain is defective, and the entire chain is masked out before running logic diagnosis. The diagnosis resolution may suffer due to masked chain(s). You should run chain diagnosis in this situation.

  Chain — A literal that enables chain diagnosis only. If the failure file contains a *chain_test* keyword and chain test failures, a chain diagnosis is performed otherwise the failure file is skipped.

  Iddq — A literal that enables IDDQ diagnosis only.

- -Verify_patterns **ON** **[Noverbose | Verbose] | OFf**

  A switch and literal pair that enables or disables verification of the external patterns specified with the diagnose_failures command. Test pattern verification is critical to accurate diagnosis results, so you should not disable it unless patterns have been previously verified.

  ON — A literal that enables verification of the external patterns. If verification is enabled, the first diagnose_failures command entered simulates the external patterns and compares the capture values with the expected values in the external patterns. When a mismatch occurs, verification stops and an error is issued. Use the report_failures -Pdet command to determine the complete set of failing patterns.

  This verification is performed only once per session for a particular set of patterns.

  ON_Noverbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool does not print out the cell pathname associated with the mismatched bit. When you use this switch, the ‘read failure’ does not produce the cell paths either.

ON Verbose — When pattern verification is enabled and a failure file mismatch occurs, then the tool prints out the cell pathname associated with the mismatched bit. This is the default.

OFf — A literal that disables verification of the external patterns.

- -Report **Default** | ***percentage***

  A switch and value pair that changes the diagnosis report generated by the tool. You have the following two report choices:

  Default — A literal that specifies to report the following for each symptom:

  - All suspects with score greater than or equal to 80.
  - Up to top three suspects even if the suspects' scores are less than 80.
  - If the suspect beyond the top three has the same score with the 3rd suspect, the tool will report the suspect only if:

    a. The suspect has same score as suspect #3, and the suspect can explain same or higher number of failing patterns.

    b. In the case the suspect has same score and fail_match, the suspect will be reported only if it has the same passing pattern mismatch.

Note

For definitions of the terms "symptom" and "suspect", refer to the *Tessent Glossary*.

percentage — An integer in the range 1 to 100 that specifies a percentage of the suspects the tool should report for each symptom. If you specify 75, for example, and there are four suspects for a particular symptom, the tool will report the top 75% (top three) of them.

- -AT_speed **ON | OFf**

  A switch and literal pair that specifies performing at-speed failure diagnosis. See "At-Speed Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information, including the conditions Tessent Diagnosis uses when diagnosing at-speed failures.

- -MAx_suspects **Default |** ***limit***

  A switch and value pair limiting the number of suspects per symptom in the diagnostics report. Choose one of the following options:

  Default — A literal that specifies reporting 100 suspects per symptom.

  limit — An integer that specifies to report this number of suspects per symptom.

- -Time_limit **OFf** | ***seconds***

  A switch and value pair that specifies a time limit, in seconds, for diagnosing a single failure file. This time includes the failure file verification time, but not the pattern verification time.

By default, the test patterns and each failure file is verified for consistency when a diagnosis is run.

If the specified time limit is reached before completion of the diagnosis, the diagnosis is aborted. A “Diagnosis aborted due to time limit (<time>)” message displays and no suspects are reported.

- -WALL_time_limit **OFf** | ***seconds***

  A switch and value pair that specifies a wall time limit, in seconds, for diagnosing a single failure file. When you specify a wall time limit (***seconds***) and a diagnosis job exceeds the limit, then the tool aborts the diagnosis run with an error message.

Note

The wall time limit is independent of the CPU time limit (using the -Time_limit switch). If you set to both the CPU and wall time limit, then the tool aborts the diagnosis when either limit is reached.

If the wall time limit is reached during diagnosis, the tool produces an incomplete diagnosis report file. If you specify the diagnose_failures command with the -csv switch, then the tool also creates an empty CSV file.

- -Pattern_sampling {***failing_pattern_limit*** | **OFf**} {***passing_pattern_limit*** | **OFf**}]

  Optional switch and value pair that specifies pattern limits for logic diagnosis. Use this switch to limit the patterns used for logic diagnosis and reduce run time. By default, all failing and passing patterns are used for diagnosis. To ensure adequate diagnosis resolution, the pattern sample must be 32 or greater.

  Logic diagnosis patterns with several capture cycles require more simulation time and have less resolution. To avoid losing resolution, Tessent Diagnosis selects patterns with the least amount of capture cycles for the diagnosis. Pattern sampling options include:

  failing_pattern_limit — Must be an integer of 32 or higher.

  passing_pattern_limit — Must be an integer of 32 or higher. Specified number of passing patterns are selected from the first passing patterns in the set.

- -FAILurefile_mismatch_verbosity **Default** | ***number*** | **All**

  A switch and value pair that controls the number of pattern mismatches displayed during the verification of the failure file. You have the following report choices:

  Default — A literal that specifies to display 20 failing patterns. This is the default.

  number — An integer that specifies to report this number of failing patterns.

  All — A literal that specifies to report all failing patterns.

- -BRidge_analysis **ON** | **OFf**

  A switch and literal pair that determines if diagnosis includes an analysis to identify bridge suspects. A bridge suspect is identified when faulty behavior is observed for the implicated

nets and the appropriate aggressor-victim conditions are met. Currently 2-way and 3-way bridges are considered and this switch is applicable to both.

Options include:

ON — Bridge analysis is enabled. Default setting.

OFf — Bridge analysis is disabled.

There are two cases where it may be desirable to disable bridge analysis.

- **Case 1** — Because the analysis is currently based only on logical responses, it is possible that a bridge suspect may not be physically possible. In cases where bridge suspects are verified as not physically possible, you may want to rerun the diagnosis with bridge analysis turned off.
- **Case 2** — In cases where very few failing patterns are captured for a failing device, and bridge analysis may return a large number of suspects, you can turn off the bridge analysis to reduce the number of suspects.

- -CHEck_last_shift_value **ON** | **OFf**

  An optional switch and literal pair that controls whether last shift values are considered for transition launching in resistive open diagnosis as well as cell internal diagnosis.

  ON — Enables last shift values consideration for transition launching in resistive open diagnosis. This is the default.

  OFf — Disables considering last shift values for transition launching in resistive open diagnosis. Use this option if you believe that the shift is very slow speed such that the transition launched from last shift is irrelevant.

- -B2X_mismatch **Warning [Noverbose | Verbose] | Error**

  A switch and literal pair that determines how B2X mismatches are handled during diagnosis. B2X mismatches occur during pattern verification when the simulator returns X values instead of the expected binary values. These errors are usually due to software enhancements to the simulator or using a mask file when reading patterns. Choose one of the following option combinations:

  Warning Noverbose — Warns at the end of pattern verification of any B2X mismatches.

  Warning Verbose — Prints during pattern verification any mismatched B2X bits.

  Error — Prints the B2X mismatches for the first mismatched pattern and terminates the run.

- -X2B_mismatch **Warning [Noverbose | Verbose] | Error**

  A switch and literal pair that determines how X2B mismatches are handled during diagnosis. X2B mismatches occur during pattern verification when the simulator returns binary values instead of the expected X values. Options include:

  Warning Noverbose — Warns at the end of pattern verification of any X2B mismatches.

Warning Verbose — Prints during pattern verification any mismatched X2B bits.

Error — Prints the X2B mismatches for the first mismatched pattern and terminates the run.

- -CELl_internal_analysis **ON** | **OFf**

  An optional switch and literal pair that determines if the tool performs analysis to diagnose defective library cells and distinguish them from defects on nets interconnecting library cells.

- -MAX_Faulty_chains *failed_chain_threshold*

  An optional switch and integer pair that specifies the maximum number of failing scan chains within a single datalog that are diagnosed using chain fault model. When a failure log has multiple failed_chains, there can either be multiple defects in scan chains, or a single defect affecting the scan control signals. The default *failed_chain_threshold* is 2.

  - If there are one or two failed_chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis. This is the default behavior of the tool.
  - If failed_chains is greater than two, then the tool assumes the defect is on a global control signal. If “set_diagnosis_options -ABORT_DIAGNOSE_MANY_faulty_chains” is OFf, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tool aborts diagnosis for this case.

  Changing *failed_chain_threshold* from the default value (2) causes the following effects:

  - If failed_chains is greater than *failed_chain_threshold*, then the tool assumes the defect is on a global control signal. If “set_diagnosis_options -ABORT_DIAGNOSE_MANY_faulty_chains” is OFf, it uses the clock/scan_enable fault model to run diagnosis. Otherwise, the tools aborts diagnosis for this case.
  - If there are one, two, ... *failed_chain_threshold* faulty chains, then the tool assumes the defect is on each individual chain and uses the scan chain fault model to run diagnosis on each of the defects.

  For example, a failure file contains six failed_chains. You can diagnose each of these six failed-chains as individual defects by setting this switch as follows:

  set_diagnosis_options -max_faulty_chains 6

Note

The run time is roughly linearly increased with the total number of failed_chains.

- -Chain_diagnosis_result **CEll** | **CHain**

  Optional switch and literal pair that determines if a preliminary high-level chain diagnosis is performed versus a standard diagnosis. A preliminary high-level chain diagnosis only reports one failure per chain; the dominant failure.

  Options include:

  CEll — Runs the standard diagnosis of chain plus scan patterns and chain plus scan failures to determine cell defects. Default setting.

  CHain — Runs a preliminary high-level diagnosis of the chain failures in the failure file and reports the name, length, and fault type of the chains that failed.

- -CHAIN_LIB_internal_pathname **OFf** | **ON**

  An optional switch and literal pair that reports the internal instance pathname within the library for each scan latch in a suspect scan cell.

  When set to ON, the tool reports this information in the chain diagnosis report's "lib_internal_pathname" column. If there is no instance name, the tool reports "". If you specify the diagnose_failures command with the -csv switch, then the tool also includes this information in the CSV output.

  See "Chain Diagnosis Section" in the *Tessent Diagnosis User's Manual* for complete information.

- -ABORT_DIAGNOSE_COMPOUND_faults **ON** | **OFf**

  An optional switch and literal pair that specifies whether or not the tool diagnosis the logic part of compound defects during chain diagnosis. The default to "ON", meaning the tool does not diagnose the logic portion of compound defects.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MANY_faulty_chains **ON** | **OFf**

  An optional switch and literal pair that instructs the tool to skip diagnosing when #faulty_chains is larger than the *failed_chain_threshold* specified with the set_diagnosis_options -MAX_Faulty_chains switch. The default is "ON".

  When you set this option to OFF and the number of faulty chains it greater than specified, the tool does not report each faulty chain.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MINIMUM_Chain_failing_probability *number*

  An optional switch and integer pair that specifies the minimum chain fail probability for the tool to abort chain diagnosis. The default is 1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -ABORT_DIAGNOSE_MINIMUM_Scan_pattern_failing_probability *number*

  An optional switch and floating point pair that specifies the minimum scan pattern fail probability for the tool to abort chain diagnosis. The default is 0.1. The minimum is 0, which means no abort. The maximum value you can specify is 100.

  See "Abort Conditions for Chain Diagnosis" in the *Tessent Diagnosis User's Manual* for a complete discussion.

- -IGNORE_TOOL_version **OFf** | **ON**

  An optional switch and literal pair that instructs the tool to check the tool version stored in the statrtup cache. If the tool version used to create the startup cache is older than the current tool version, Tessent Diagnosis does not load the startup cache because the simulation results may have changed.

  Only set this option when you load or update the startup cache, not when you create the cache.

  When set to ON, Tessent Diagnosis disables the tool version check and uses the startup cache regardless of its tool version.

- -INCLude_fail_signatures_size *maximum_rows_per_table* | MAX

  An optional switch and integer pair that specifies the maximum number of rows per table for reporting failure signature information in the diagnosis report. The default is 256. If you do not want the failure signatures included in the diagnosis report, then specify 0 (zero) with this switch. Conversely, if you require all signature information, then use the keyword MAX with this switch. See "Failure Signature Information in the Diagnosis Report" in the *Tessent Diagnosis User's Manual* for complete information.

- -INCLUDE_BRIDGE_to_power ON | OFf

  An optional switch and literal pair that specifies performing analysis and reporting of potential bridging defects between the STUCK suspect net, and a power or ground line during layout-aware diagnosis. The default is "ON".

  See "Power and Ground Bridge Reporting" in the *Tessent Diagnosis User's Manual* for complete information.

- -GRoss_delay ON | OFf

  An optional switch and literal pair that specifies performing analysis and reporting of gross delay defects. The default is "OFf".

  See "Gross Delay Defect Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

- -COMPOUND_Hold_time_fault_diagnosis **ON** | **OFf**

  An optional switch and literal pair that specifies performing analysis and reporting of slow clock compound hold-time defects. The default is "OFf".

  See "Slow Clock Compound Hold-Time Diagnosis" in the *Tessent Diagnosis User's Manual* for complete information.

- -INCLUDE_DFM_rules **ON** | **OFf**

  An optional switch and literal pair that specifies to perform DFM hit reporting. The default is “ON”.

  See “Diagnosis for Design for Manufacturability Analysis” in the *Tessent Diagnosis User’s Manual*.

- -INCLUDE_RCD_constants **ON** | **OFf**

  An optional switch and literal pair that specifies to populate the LDB with RCD constants. The default is “ON”.

  See “Diagnosis for Root Cause Deconvolution Analysis” in the *Tessent Diagnosis User’s Manual*.

- -EXPECTED_value **ON** | **OFf**

  An optional switch and literal pair that specifies whether to verify expected pattern values against actual pattern values and return the results in the failure log. When set to OFF, the tool suppresses this check and ignores any missing expected values.

Caution

The -expected_value switch turns off failure log verification, which means that you will not know whether there is a discrepancy between the failure log and the patterns. Use this switch with extreme caution.

- -internal_failing_cells **ON** | **OFf**

  An optional switch and literal pair that specifies whether to enable internal scan cell profiling. See “Internal Scan Cell Profiling for Compressed Patterns” in the *Tessent Diagnosis User’s Manual*.

- -cycle_offset *N*

  An optional switch and integer pair that specifies to adjust the cycles in a cycle-based failure file by the value *N*. Tessent Diagnosis uses the adjusted cycles for failure verification and diagnosis. See “Cycle Offset Adjustment for Failure Files” in the *Tessent Diagnosis User’s Manual*.

Example 1

The following example loads patterns from two files into the external pattern database for the tool. The failure file, *scan_failure_file*, contains just a scan_test section, so the tool only performs a scan diagnosis for that file.

```
read_patterns pat_file1
read_patterns pat_file2 -append
set_diagnosis_options -mode scan
diagnose_failures scan_failure_file -output scan_diag.ascii -csv scan_diag.csv
```

Example 2

The following example runs a preliminary diagnosis on my_failure_file.

read_patterns pat_file1

set_diagnosis_options -chain_diagnosis_result chain

diagnose_failures my_failure_file

Example 3

The following example shows a preliminary diagnosis report.

```
tracking_info_begin
tracking_info_end
#faulty_chains=2 #symptoms=0 #suspects=0 CPU_time=0.00sec
fail_log=fail_logs/my_failure_file
Chain chain1 has error, fault type is STUCK_AT_0, the length of this chain
is 102.
Chain chain3 has error, fault type is STUCK_AT_1, the length of this chain
is 101.
```

Related Commands

diagnose_failures
report_failures
write_failures

set_display

Context: unspecified, all contexts

Mode: all modes

Usage

set_display ***display_name***

Description

Sets the DISPLAY environment variable from the tool's command line.

The set_display command sets the DISPLAY environment variable to display_name without exiting the currently running application. If you invoke the tool in command line mode (the default), then the DISPLAY variable is not required in order to use most commands successfully. If the variable is not set, however, and you issue a command such as open_visualizer that requires a GUI, the tool will not be able to show the applicable window. This is because the display connection provided by the variable is absent. The set_display command enables you to set the variable, if needed for the display, without the inconvenience of exiting to the shell in order to set it.

Note

This command affects the DISPLAY setting within the currently running application only. When you exit the tool, the setting in the invocation shell will be what it was when you invoked the tool.

Arguments

- ***display_name***

  A required string that specifies a valid display setting for the machine on which the tool is running.

Example

The following example sets the DISPLAY variable when "open_visualizer" fails to open DFTVisualizer. The example also uses the system command to pass a Linux "echo $DISPLAY" command to the shell in order to check the variable's setting.

open_visualizer

```
//============================================================
// Warning: The DFTVisualizer window(s) will not be accessible
// for this session due to the lack of a valid X display
// connection.
//
//       Try using the set_display command to set your DISPLAY
//       variable.  OR start/restart your Xserver.
//============================================================
```

```
system echo $DISPLAY

set_display my_workstation:0.0
system echo $DISPLAY
```

```
my_workstation:0.0
```

Related Commands

system

set_dofile_abort

Context: unspecified, all contexts

Mode: all modes

Usage

set_dofile_abort **ON** | **OFf** | **exit**

Description

Specifies whether the tool aborts or continues dofile execution if it detects an error condition.

By default, if an error occurs during the execution of a dofile, processing stops, and the line number causing the error in the dofile is reported. The set_dofile_abort command lets you turn this functionality off so that the tool continues to process all commands in the dofile. In this case, the tool aborts the current Tcl construct (if/for/while) but continues to process the remaining commands in the dofile. You can use the Tcl catch command or the Tessent catch_output command to manage errors and prevent the tool from aborting the current construct.

Arguments

- **ON**

  A literal that halts the execution of a dofile upon the detection of an error. When in batch mode the tool exits on error, but when in interactive mode the tool returns to the session prompt. This is the default upon invocation of the tool.

- **OFf**

  A literal that forces dofile processing to complete commands in a dofile regardless of error detection. If an error is encountered in a Tcl if, for, or while construct within the dofile file, the tool exits the construct immediately but continues execution of the next sequential command outside the construct. Note, this option does not affect the Tcl source command which completely stops execution of the source file when an error is encountered.

- **exit**

  A literal that directs the tool to exit the session if it detects an error while executing a dofile regardless of whether invoked in batch or interactive mode.

Example 1

The following example sets the set_dofile_abort command off to ensure that all commands in *test1.dofile* are executed.

```
set_dofile_abort off
dofile test1.dofile
```

Example 2

In the following example dofile, if cmd1 returns an error and set_dofile_abort is set to On, cmd2 does not execute and the tool exits the dofile.

If cmd1 returns an error and set_dofile_abort is set to Off, cmd2 still does not execute but the tool processes the remainder of the dofile. You can use the Tcl catch command or the tessent catch_output command around cmd1 to manage the errors generated by cmd1 and allow the tool to continue with cmd2 even if cmd1 generated an error. The set_dofile_abort command only affects how the error is handled when it propagates up the nested command stack and reaches the dofile level.

```
if {1}  {
  cmd1
  cmd2
}
if {1}  {
  cmd3
  cmd4
}
```

Related Commands

catch_output

dofile

set_drc_handling

Context: unspecified, all contexts

Mode: all modes

Usage

set_drc_handling {***drc_id*** [Error | Warning | NOTe | Ignore]
[NOVerbose | Verbose] [NOAtpg_analysis | Atpg_analysis]
[ATPGC] [-Mode A *clk_name*] [-Interval *number*] [-CONSERvative {ON | OFf}]
[-SKip_procedure {TEst_setup | OFf}]
[-TIMing_exception_check { ON | OFf }]
[-Mode {Sequential | Combinational}]
[-SIMULATE_ACROSS_Internal_primary_inputs { ON | OFf }] }
| -INItialization_cycle {ON | OFf}
| -ALLOW_COMmon_scan_cells_in_multiple_chains { ON | OFf }
| -SCan_chain_tracing {Aggressive | Conservative}

Description

Specifies how the tool handles design rule violations.

The set_drc_handling command specifies the handling of the messages for the following design rule checks:

- scan cell RAM rules checking (A rules)
- some BIST rules checking (B2 and B4 rules)
- Clock rules checking (C rules)
- Data rules checking (D rules)
- Extra rules checking (E rules)
- EDT Finder rules checking (F rules)
- EDT rules checking (K rules)
- Procedure rules checking (P rules)
- some Scan Pattern Retargeting rules checking (R rules)
- Trace rules checking (T rules)
- some Power-aware rules checking (V rules)
- certain Timing rules checking (W rules)

You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses either the handling from the last set_drc_handling command or, if you did not change the handling, the Design Rules Checker‘s invocation default.

Note

The set_drc_handling command does not support the scannability (S) rules except for the S3 and S5 rules. Unlike the other DRC rules, setting the handling type to “Error” does not turn on the transcript verbosity for the S3 rule. The verbosity can only be turned on via the optional literal “verbose”.

The set_drc_handling command does not support any Flattening design rules (FN, FP, and FG rules). Use the set_flattener_rule_handling command to specify how design rule violations are handled for the Flattening rules.

The set_drc_handling command does not support the K15 rule.

Each rules violation has an associated occurrence message and summary message. The tool only displays the occurrence message for either error conditions or if you specify the Verbose option for that rule. The tool also displays the rule identification number in all rules violation messages.

The Atpg_analysis option provides test generation analysis when performing rules checking for some clock (C) rules, for some data (D) rules, and for some extra (E) rules. For example, if Atpg_analysis is enabled for clock rule C1, and the tool simulates a clock input as X, the rule violation occurs when it is possible for the test generator to create a test pattern while that clock input is on, all defined clocks are off and all constrained pins are at their constrained state.

Note

When Atpg_analysis is enabled, the tool requires some additional CPU time and memory to perform the test generation analysis. (The Atpg_analysis option is enabled by default for rules C1, E10, E11 and E13; you can disable it for these rules by specifying the Noatpg_analysis option.)

Arguments

- ***drc_id***

  A required non-repeatable literal that specifies the identification of the exact design rule violation whose message handling you want to change.

  For more information on the design rules, refer to “Design Rule Checking”.

- Error

  An optional literal that specifies for the tool to both display the error occurrence message and immediately terminate the rules checking.

- Warning

  An optional literal that specifies for the tool to display the warning summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rules violation.

- NOTe

  An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation

- Ignore

  An optional literal that specifies for the tool not to display any message for the rule's violations. The tool must still check some rules and they must pass to allow later performance of certain functions.

- NOVerbose

  An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display a summary message indicating the total number of occurrences of the rule violation. This is the default.

  If DRC handling is set to Error or Ignore, the tool will behave as follows regardless of whether you specify the Noverbose (or Verbose) option:

  - Error—The tool will stop at each error and display detailed instance-specific violation information to help you debug it.
  - Ignore—The tool will not perform the check and therefore, the consequences of the check on ATPG, whether good or bad, will not occur. Also, the tool will not display or store instance-specific violation information.

- Verbose

  An optional literal that, for a DRC whose handling is set to Warning or Note, specifies to display the occurrence message for each occurrence of the rule violation. All instance-specific violations (maybe hundreds or thousands) will be listed, one message per instance violating the rule. To obtain a summary message instead, use the Noverbose option.

  If the DRC's handling is set to Error or Ignore, the tool will behave as follows regardless of whether you specify the Verbose (or Noverbose) option:

  - Error—The tool will stop at each error and display detailed instance-specific violation information to help you debug it.
  - Ignore—The tool will not perform the check and therefore, the consequences of the check on ATPG, whether good or bad, will not occur. Also, the tool will not display or store instance-specific violation information.

- NOAtpg_analysis

  An optional literal that specifies for the tool not to use test generation analysis when performing rules checking. This is the default except for rules C1, E4, E10, E11 and E13.

- Atpg_analysis

  An optional literal that specifies for the tool to use test generation analysis when performing rules checking for clock rules (such as C1, C3, C4, C5 and C6), some D rules (such as D6

and D9) and some E rules (such as E4, E5, E8, E10, E11 and E12). This is the default for rules C1, E4, E10, E11 and E13.

For clock rules C3 and C4, the Atpg_analysis option generates a check of the clocks of the source and sink to see if they are gated off.

Note

To use the constraint values during the D6 rule analysis, you must use the Atpg_analysis option.

- ATPGC

  An optional literal that specifies for the design rules checker to use all the current ATPG constraints when analyzing E10 rule violations. You can also use the "add_atpg_constraints -Static" command to do the same thing.

- -Mode A *clk_name*

  A switch, a literal (A), and a string triplet that specifies the name of a clock on which you want the tool to perform further analysis to screen out false C3 and C4 clock rules violations. The tool performs this extra analysis on one clock only.

  For more information on using the -Mode option, refer to the C3 and C4 descriptions.

- -Interval *number*

  An optional switch and integer pair that you can only use with C3 and C4 clock violations to specify how often you want the tool to display a message during the ATPG analysis of those violations. The *number* argument indicates multiples of violation occurrences that cause the tool to display a message. The default is 0.

  The message includes the number of sequential elements that the tool checked, the number of sequential elements remaining to check, the current number of ATPG passes during the C3 or C4 clock rules checking, and the current CPU time used by the tool for clock rules checking.

  The value of the *number* parameter must be either zero or a positive integer. You can only specify one *number* value that the tool uses for both the C3 and C4 violations. If you issue multiple set_drc_handling commands (one for C3 and one for C4) that specify different values for the *number* argument, the tool uses the last interval value you specified.

- -CONSERvative {ON | OFf}

  An optional switch and literal for C6 DRC analysis. The default if OFF.

  The default C6 checking may miss some C6 violations when set_clock_restriction OFF is used. A more conservative analysis can be enabled by setting the -CONSERvative to ON and will identify those C6 violations. However, it is generally recommended not to turn off clock restriction such that the -CONSERvative switch does not usually need to be used.

- -SKip_procedure {TEst_setup | OFf}

  An optional switch and literal pair that affects the tool's handling of the E4 rule. The default option specifies to skip these checks when simulating the test_setup procedure. The tool will continue to check for contention during other test procedures. The Off option specifies for the tool to perform E4 rule checks for the test_setup procedure, as well as other procedures, during DRC.

  It is very common during test_setup, when a design is being initialized, to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention, and the tool may issue many E4 violation messages. If you wish to be notified about this type of bus contention during test_setup, use the Off option to include test_setup for E4 rule checks. This switch has no effect if you have previously set E4 handling to Ignore.

- -TIMing_exception_check { ON | OFf }

  An optional switch and literal that enables the timing exception check for the E5 DRC. The default is off, in which case the tool does not consider timing exception paths during E5 DRC (checking for X sources). See “Example 8” on page 1278.

- -Mode {Combinational | Sequential}

  An optional switch and literal for the E10 rule.

  Combinational — Default upon invocation. This option performs bus contention mutual-exclusivity checking. This checking differs from rule E4 in that it does not check for this condition during test procedures.

  Sequential — Considers the inputs to a single level of sequential cells behaving as “staging” latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enables of the tri-state devices except through the sequential cells.

- -SIMULATE_ACROSS_Internal_primary_inputs { ON | OFf }

  An optional switch and literal pair that changes how the tool simulates internal PI’s during DRC. (Internal PI’s are primary inputs you previously defined using “add_primary_inputs -internal” or “add_clocks -internal.”)

  After you set this switch to ON in setup mode, during test procedure simulation the tool starts off simulating an internal PI as an internal signal by carrying over its driver’s simulation value until the point when you first supply a simulation value by explicitly forcing this PI in any of the following test procedures: test_setup, load_unload, or shift.

  This switch is needed for cases where you may have a cut point (such as for capture modeling), yet you need test_setup to be simulated normally as if the cut point were not present. This is commonly needed when using IJTAG within test_setup, where you want the IJTAG network simulated normally even if the network includes a cut point. For more

information about using IJTAG and test procedures, refer to "Test Setup and Test End Procedures" in the *Tessent IJTAG User's Manual*.

- -INItialization_cycle {ON | OFf}

  An optional switch and literal pair that controls whether the tool forces all clocks to their off states and all constrained pins to their constrained values prior to simulation of the test_setup procedure during DRC. By default, the tool performs these forces, so you do not have to explicitly specify the forces, which matches the default initialization cycle the tool includes in saved patterns. If you have occasion to write patterns without this initialization cycle (write_patterns -Noinitialization), issue a set_drc_handling -Initialization Off command and rerun DRC before saving patterns.

Tip: If DRC is run with initialization enabled, turning off initialization when saving patterns (write_patterns -Noinitialization) may result in simulation mismatches during pattern verification in a timing based simulator. The converse (DRC run with initialization disabled and patterns saved with initialization enabled) is typically not a problem.

The argument choices are as follows:

ON — A literal that specifies to initialize the clocks and constrained pins to their off states and constrained values, respectively, prior to DRC simulation of the test_setup procedure. This is the invocation default.

OFf — A literal that specifies not to initialize the clocks and constrained pins prior to DRC simulation of the test_setup procedure.

- -ALLOW_COMmon_scan_cells_in_multiple_chains { ON | OFf }

  An optional switch and literal pair that specify whether or not to allow scan cells to be shared among scan chains. Note that there are some restrictions on scan cell sharing, which result in a T2 violation when not observed:

  - Scan chain sharing is not allowed for compressed chains driven by a decompressor.
  - The shared scan cells must be at the logical scan cell boundary. A library model with multiple scan cells cannot be partially shared. In other words, you cannot have the merged point within the state elements of a logical scan cell.

- -SCan_chain_tracing {Aggressive | Conservative}

  An optional switch and literal pair that specify whether or not scan chain tracing passes with a T25 violation. The choices are as follows:

  Aggressive — Scan chain tracing passes with a T25 violation. This is the default. For more information on T25 violations, see the "T25" section.

  Conservative — Scan chain tracing does not pass with a T25 violation. In this case, scan chain tracing fails, design rule checking stops, and the tool issues a T3 violation. For information on correcting T3 violations, see the "T3" section.

Example

The following example specifies rule checking E4 to be an error:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
set_drc_handling e4 error
set_system_mode analysis
```

Related Commands

report_drc_rules

set_driver_restriction

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

Mode: setup, analysis

Usage

set_driver_restriction **OFf** | **ON**

Description

Specifies whether the tool allows multiple drivers on buses and multiple active ports on gates.

The set_driver_restriction command lets you specify for the tool to report a contention problem whenever there are multiple nets driving values onto a bus, or when multiple ports are active on an individual gate. The default upon tool invocation is to allow multiple driving nets on a bus or multiple active ports on a gate as long as the signals are driving the same value. If multiple signals are on and are not driving the same values, then the tool flags it as contention.

However some design processes only allow a single driver to be on at a time regardless of whether the signals are driving the same values. The set_driver_restriction command lets you place this same restriction on the tool.

Arguments

- **OFf**

  A literal that specifies to allow multiple drivers to be on for a bus and multiple ports to be active for a gate as long as the driving signals are of the same value, and therefore there is no contention. This is the default behavior when you invoke the tool.

- **ON**

  A literal that restricts buses to only have one driver on at a time and gates (dff and latches with multiple clocks) to only have one active port; the tool flags multiple active drivers or ports as contention problems.

Example

The following example creates a strict contention checking environment. The first command specifies for the tool to check for contention on both multiple port gates and buses. If there is a contention problem where signals are driving different values, the tool reports an error and stops the simulation. The second command further restricts the contention checking environment by not allowing multiple drivers to even drive the same values.

```
set_contention_check on -error -all
set_driver_restriction on
```

Related Commands

set_contention_check

set_edt_finder

Context: patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

set_edt_finder [OFf | ON] [-Verbose {OFf | ON}]

Description

Enables the tool to automatically identify EDT logic and update scan chain pins during test pattern generation.

This command can be used in conjunction with Core Mapping for ATPG. For more information, see "Core Mapping for Top-Level ATPG" in the *Tessent Scan and ATPG User's Manual*.

You can use this command to get updated scan pin information when your design hierarchy changes after the EDT logic is generated. For more information, see "Updating Scan Pins for Test Pattern Generation" in the *Tessent TestKompress User's Manual*.

If no arguments are specified, the current EDT Finder settings display.

Note

The EDT Finder must be enabled before any scan chains are defined for ATPG.

Note

When the EDT Finder is enabled, DI faults are automatically identified in EDT logic. You can disable this behavior by executing the following command:

```
set_fault_subclass_analysis -disable DI.EDT
```

For more information, see "Fault Sub-classes" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- OFf

  An optional literal that disables EDT Finder. This is the default.

- ON

  An optional literal that enables EDT Finder.

- -Verbose {OFf | ON}

  An optional switch and literal pair that enables the transcription of informational messages about the EDT logic found. The transcription of informational messages is disabled upon invocation of the tool.

Example 1

The following example shows the output of the EDT Finder for a design with two channels and eight internal scan chains. The EDT Finder output is highlighted in bold.

```
set_edt_finder on
dofile results/generated_edt.dofile
add_scan_groups grp1 results/generated_edt.testproc
add_scan_chains -internal chain1 grp1 /top_i/si1 /top_i/so1

add_scan_chains -internal chain2 grp1 /top_i/si2 /top_i/so2
set_edt_pins update EdtUpdate
set_edt_pins input_channel  1 ChannelInput_1
set_edt_pins output_channel 1 ChannelOutput_1
set_edt_pins input_channel  2 ChannelInput_2
set_edt_pins output_channel 2 ChannelOutput_2
set_mask_decoder_connection -mode_bit 1 5
...
set_system_mode analysis
```

```
// Flattening process completed,  design_cells=221  leaf_cells=221
library_primitives=337  sim_gates=471  PIs=16  POs=11  CPU time=0.00 sec.
//  ----------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ----------------------------------------------------------------------
...
//  ----------------------------------------------------------------------
//  Begin EDT finder analyses.
//  ----------------------------------------------------------------------
//  EDT Finder completed, EDT blocks=1, scan chains=8, CPU time=0.02 sec.
//  ----------------------------------------------------------------------
//  Begin scan chain identification process, memory elements = 54.
//  ----------------------------------------------------------------------
...
```

Related Topics

report_edt_finder

set_edt_instances

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_edt_instances [-Edt_logic_top *edt_instance_name_or_pathname*]
[-Compactor *compactor_instance_name_or_pathname*]
[-Decompressor *decompressor_instance_name_or_pathname*]
[-Block_location *path_to_top_of_block*]

Description

IP Creation phase:

Specifies the location in which the tool places the EDT logic.

In the IP Creation phase, you use this command with the -Block_location switch to specify the parent instance in which the EDT logic is to be inserted. See Example 1. Note, if you use any switch except -Block_location in this phase, the tool ignores the switch and issues a warning.

By default, the EDT logic is instantiated at the top level of the design.

Pattern Generation phase:

Specifies an instance name/location for existing EDT logic or for the design block that contains the EDT logic (EDT block).

You can use this command in the Pattern Generation phase to:

- Tell the tool the new name of an already renamed EDT logic block, decompressor, or compressor. See Example 3.
- Name each EDT block uniquely, using the -Block_location switch, when an EDT block is used multiple times. See Example 4.

  The set_edt_instances command operates only on the current EDT block which is the last block specified by either the add_edt_blocks or set_current_edt_block commands. For more information, see "Modular Compressed ATPG" in the *Tessent TestKompress User's Manual*.

Arguments

- -Edt_logic_top *edt_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the design block containing the EDT logic (EDT block). Names specified with this switch must be unique. If the specified edt_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- -Compactor *compactor_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the compactor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified compactor_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- -Decompressor *decompressor_instance_name_or_pathname*

  A required switch and string pair that specifies an instance name or pathname for the decompressor sub-block within the current EDT block. Names specified with this switch must be unique. If the specified decompressor_instance_name_or_pathname is not unique in the design, the tool issues a message; in this case, you can specify the full pathname.

- Block_location *path_to_top_of_block*

  A required switch and string pair that specifies the location of the current EDT block within the top-level netlist.

  During EDT IP Creation, use this switch to specify the location in which the tool places the EDT logic.

  During Pattern Generation, use this switch to identify each EDT block uniquely when an EDT block is used multiple times, by specifying its parent instance pathname.

  For more information, see "Modular Tessent TestKompress" in the *Tessent TestKompress User's Manual*.

Example 1

The following command is issued during EDT logic creation to insert the EDT logic in the instance, pad_east/test_blk, instead of at the top level of the design:

set_edt_instances -block_location pad_east/test_blk

Example 2

The following example specifies the name of the instance that contains the EDT logic:

set_edt_instances -edt_logic_top my_core_edt_i

```
// Found edt_logic_top at path: "/my_core_edt_top/my_core_edt_i".
```

Example 3

The following example specifies the instance names of the decompressor and compactor for the EDT logic and reports the current instance locations:

**set_edt_instances -decompressor my_core_edt_decompressor_i **
-compactor my_core_edt_compactor_i

```
// Found compactor at path:
      "/my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i".
// Found decompressor at path:
      "/my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i".
```

report_edt_instances

```
// edt_logic_top
```

```
//    Setting: my_core_edt_i
//    Path   : /my_core_edt_top/my_core_edt_i
//
// decompressor
//    Setting: my_core_edt_decompressor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_decompressor_i
//
// compactor
//    Setting: my_core_edt_compactor_i
//    Path   : /my_core_edt_top/my_core_edt_i/my_core_edt_compactor_i
```

Example 4

The following example specifies unique names for instances of the same EDT block in a top-level modular netlist:

set_edt_mapping on
add_edt_blocks my_core1_a

```
// my_core1_a is the current EDT block.
```

set_edt_instances -block_location /my_core1_a_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

```
// edt_logic_top
//    Setting: my_core1_edt_i
//    Path   : /my_core1_a_edt_i/my_core1_edt_i
...
```

add_edt_blocks my_core1_b

```
// my_core1_b is the current EDT block.
```

set_edt_instances -block_location /my_core1_b_edt_i
dofile my_core1_edt.dofile
...
report_edt_instances

```
// edt_logic_top
//    Setting: my_core1_edt_i
//    Path   : /my_core1_b_edt_i/my_core1_edt_i
...
```

The set_edt_mapping command enables the requisite mapping of block-level dofile commands to the top-level.

Related Commands

add_edt_blocks
report_edt_instances
set_current_edt_block
set_edt_mapping

set_edt_mapping

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

set_edt_mapping {**OFf** | **ON**} [-Verbose {**OFf** | **ON**}]

Description

Enables or disables automatic mapping of block-level dofile commands so block-level dofiles can be reused without modification for top-level pattern creation.

Note

This command and the EDT Mapping functionality have been superseded by the newer Core Mapping for ATPG feature. Please use the newer capability described in "Core Mapping for Top-Level ATPG" in the *Tessent Scan and ATPG User's Manual*.

If you are using the EDT integration flow that uses the add_edt_connections command to manually create the connections from the blocks to the top level, you can continue to use the set_edt_mapping command and EDT Mapping functionality. In this case, you can still use the Core Mapping for ATPG feature for transferring core information to the top level for pattern generation.

The set_edt_mapping command determines whether or not the tool performs the automatic mapping necessary to reuse, for top-level pattern generation, the dofiles the tool generated during block-level IP creation. You must set this command to "on" prior to running block-level dofiles to set up for top-level pattern creation.

For more information about the modular Tessent TestKompress flow or the reuse of block-level dofiles, refer to the "Modular Compressed ATPG" chapter of the *Tessent TestKompress User's Manual*.

Note

Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_edt_mapping commands until you specify a different setting

Arguments

- **OFf**

  A literal that turns off automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase. This is the default upon invocation of the tool.

- **ON**

  A literal that enables the automatic mapping necessary to reuse the block level dofiles generated during the IP Creation Phase.

- -Verbose {**OFf** | **ON**}

  An optional switch and literal pair that enables or disables the tool's transcription of informational messages whenever it changes any of the following block-level dofile objects as a result of automatic mapping:

  scan group name
  scan chain name
  pin pathname
  pin name

  The transcription of informational messages is disabled upon invocation of the tool.

Example 1

The following example enables the automatic mapping of commands from the block-level dofiles to the top-level design hierarchy, then executes the commands in one of the block-level dofiles:

set_edt_mapping on

delete_edt_blocks -all

```
// All blocks deleted.  No EDT block now selected as current block.
```

add_edt_blocks created_core1

```
// created_core1 is the current EDT block.
```

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile

```
// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Will use top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// command: add_clocks 0 SCLK
// command: add_clocks 0 CCLK
...
```

Example 2

The following example shows the transcription that would have occurred had the -Verbose switch been in effect for the commands of the preceding example:

set_edt_mapping on -verbose on

delete_edt_blocks -all

```
// All blocks deleted.  No EDT block now selected as current block.
```

add_edt_block created_core1

```
// created_core1 is the current EDT block.
```

set_edt_instances -block_location /x1
dofile created_core1_edt.dofile

```
// command: add_scan_groups grp1 ./edt_ip/created_core1_edt.testproc
// Note: Ignoring "add_scan_groups" command for grp1. Will use top-level
// definition since scan group mapping is active.
// command: add_scan_chains -internal chain1 grp1 /m8_i/scan_in1
// /m8_i/scan_out1
// Note: Block level chain name (chain1) automatically translated to
// created_core1_chain1.
// Note: Block level path (/m8_i/scan_in1) automatically translated to
// /x1/m8_i/scan_in1.
// Note: Block level path (/m8_i/scan_out1) automatically translated to
// /x1/m8_i/scan_out1.
...
// command: add_scan_chains -internal chain4 grp1 /m8_i/scan_in4
// /m8_i/scan_out4
// Note: Block level chain name (chain4) automatically translated to
// created_core1_chain4.
// Note: Block level path (/m8_i/scan_in4) automatically translated to
// /x1/m8_i/scan_in4.
// Note: Block level path (/m8_i/scan_out4) automatically translated to
// /x1/m8_i/scan_out4.
// command: add_clocks 0 SCLK
// Note: Block level "add_clocks" command for SCLK will be verified
// during DRC.
// command: add_clocks 0 CCLK
// Note: Block level "add_clocks" command for CCLK will be verified
// during DRC.
```

Related Commands

add_edt_blocks
delete_edt_blocks
set_edt_instances

set_edt_options

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

Mode: setup, analysis (dft -edt, dft -scan, and patterns -scan contexts only)

Usage

set_edt_options [OFf | ON | Auto]
[{-CHannels ***integer***} | {-INPut_CHannels *integer*}{-OUtput_CHannels *integer*}]
[-LOCAtion {External | Internal}]
[-BYPASS_Chains *integer*]
[-OVERShift_cycles *integer*]
[-SIngle_bypass_chain {ON | OFf}]
[-BYpass_logic {ON | OFf | Use_existing_bypass_chains}]
[-CLocking {Edge | Level}]
[-PIpeline_logic_levels_in_compactor {OFf | *integer*}]
[-LOCKup_cells {ON | OFf}]
[-LONgest_chain_range {*min_number_cells max_number_cells*}]
[-RESet_signal {OFf | Asynchronous}] [-COMpactor_type {XPress | BAsic}]
[-PULse_EDt_Before_shift_clocks {ON | OFf}]
[-FOrce_1hot_masking {OFf | ON}]
[-RETIme_chain_boundaries {ON | OFf}]
[-SEParate_control_data_channels ON | OFf]

For dft -scan

set_edt_options [-Location {External | Internal}]
[-Channels ***integer***]

For Hybrid TK/LBIST Flow IP Creation

set_edt_options **-lbist_misr_input_ratio *integer* -reset_signal asynchronous**

For Hybrid TK/LBIST Flow Fault Simulation

set_edt_options [-decompressor_seed ***hex_string***]

Description

Determines whether compression is on and specifies parameters for the EDT logic, including scan channels, bypass circuitry, lockup cells, pipeline stages, and compactor type.

To create EDT logic for a basic application, you only need to specify the number of channels.

When set to Off, the tool can be used for ATPG and behaves the same as Tessent FastScan in command-line mode.

Use the report_environment command to display the current settings.

Settings are retained over multiple set_edt_options commands until a different setting is specified.

Hybrid TK/LBIST Flow Specifics

Sets options for EDT IP creation and LogicBIST fault simulation.

IP Creation: This command specifies options specific to the EDT/LogicBIST blocks.

Fault Simulation: This command describes the initial seed of the PRPG.

Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- OFf

  An optional literal that disables compression (EDT) mode and enables analysis mode .

- ON

  An optional literal that enables compression (EDT) mode.

- Auto

  An optional literal that enables automatic detection of EDT mode (compression). Note, this option is only available in the patterns -scan context.

- -CHannels ***integer***

  An optional switch and integer pair that specifies the number of external scan channels. Use this switch when the number of input channels is the same as the number of output channels.

  Use this switch to set up the number of scan channels the EDT logic is configured with. During pattern generation, this switch is used to specify the number of scan channels the EDT logic contains. The dofile created when the EDT logic is generated, contains a set_edt_options command that specifies the number of channels for test pattern generation.

- -INPut_CHannels ***integer***

  An optional switch and integer pair that specifies the number of scan input channels. Use this switch only if the number of input channels differs from the number of output channels.

- -OUtput_CHannels ***integer***

  An optional switch and integer pair that specifies the number of scan output channels. Use this switch only if the number of output channels differs from the number of input channels.

- -BYPASS_Chains ***integer***

  An optional switch and integer pair that determines how many scan chains are created for bypass mode. The specified value must be equal to or less than the number of bypass chains created by default. By default, the number of bypass chains created equals the number of input/output channels. If the number of input and output channels differ, the smaller number is used.

  For dual compression configuration applications, this switch must be issued after both configurations are defined.

- -OVERShift_cycles ***integer***

  An optional switch and integer pair that adds a specified number of shift cycles to a test pattern set. Use this switch to balance out core test pattern sets when integrating them at the chip level in the SoC flow.

  This switch can be used before saving out a particular pattern set to adjust the number of shift cycles. This is typically useful for mapping core level patterns to the chip level in the SoC pattern mapping flow. During core level pattern generation, the patterns can be saved as is. While integrating core level patterns at the chip level, the differences in the number of shift cycles for the different pattern sets corresponding to the different cores will be known. The switch can then be used to add the additional cycles to the pattern set with lesser number of cycles to balance out all the pattern sets. This facilitates pattern merging at the chip level.

- -SIngle_bypass_chain {**ON** | **OFf**}

  An optional switch and literal pair that creates compression logic with a dual bypass chain configuration. Use this option to create a bypass configuration that concatenates all scan chains together into one bypass chain in addition to the default bypass configuration. For more information, see "Dual Bypass Configurations" in the *Tessent TestKompress User's Manual*.

- -LOCAtion {External | Internal}

  An optional switch and literal pair that specifies a location for the EDT logic relative to the netlist read in. Options include:

  External — Creates a new top level "wrapper" module and instantiates both the design netlist read in and the EDT logic within it. The logic is thus kept external to the design netlist. This is the default.

  Internal — Instantiates the EDT logic in the existing top level of the design netlist.

- -BYpass_logic{**ON** | **OFf** | **Use_existing_bypass_chains**}

  An optional switch and literal pair that specifies a location for the bypass logic. Bypass logic allows you to bypass the compression circuity and run traditional ATPG with a tool such as Tessent FastScan. Bypass logic is transparent during normal operation. Options include:

  ON — Includes bypass logic in the EDT logic. This is the default.

  OFf — No bypass logic is generated.

  Use_existing_bypass_chains — Uses bypass logic already in the core logic.

- -CLocking {**Edge** | **Level**}

  An optional switch and literal pair that determines what type of architecture logic is used for the EDT logic. Options include:

  Edge — Specifies DFF-based logic. You can use DFF-based logic for designs that are LSSD-based, mux-DFF-based, or that incorporate a mixture of these two architectures. This is the invocation default.

  Level — Specifies latch-based logic. This option is for pure LSSD designs only.

- -PIpeline_logic_levels_in_compactor {**OFf** | ***integer***}

  An optional switch and literal pair that enables pipeline stages in the compactor of the EDT logic. Options include:

  OFf — Disables pipeline stages in the EDT logic. This is the default.

  integer — Includes pipeline stages in the EDT logic compactor(s), and specifies the maximum number of logic levels (2-input XOR gates) allowed in the compactors before inserting a pipeline stage.

- -LOCKup_cells {**ON** | **OFf**}

  An optional switch and literal pair that determines whether lockup cells are inserted in the EDT logic.

Caution

You should leave this switch set to *on*. Disabling lockup cell synthesis can lead to DRC violations in the pattern generation phase and inability to generate valid patterns. For more information, see "Understanding Lockup Cells" in the *Tessent TestKompress User's Manual*.

Options include:

ON — Includes lockup cells in the EDT logic as needed between the decompressor and the core, between the core and the compactors (if pipelining is used), and between scan chains when they are concatenated for bypass mode. This is the default setting.

OFf — Disables lockup cells in the EDT logic.

- -LONgest_chain_range ***min_number_cells max_number_cells***

  An optional switch and integer pair that defines a range for the length of the longest scan chain. Use this switch to avoid having to regenerate the logic for minor changes in the length of the chains after the logic is created.

Note

This switch should be used when creating EDT logic with the skeleton flow. For more information, see "Longest Scan Chain Range Estimate" in the *Tessent TestKompress User's Manual*.

Options include:

min_number_cells — Specifies the minimum number of cells in the longest scan chain. Using a reasonable estimate of the lower bound of the length of the longest scan chain prevents the creation of several short mask registers instead of one (or a few) long mask register. More registers increase the chance of a K22 DRC violation and may lead to a higher pattern count.

max_number_cells — Specifies the maximum number of cells in the longest scan chain. A too large max_number_cells value may increase the decompressor size a little.

- -RESet_signal {**OFf** | **Asynchronous**}

  An optional switch and literal pair that specifies whether an asynchronous reset signal is included in the EDT logic. Options include:

  OFf — Disables the asynchronous reset. This is the default.

  Asynchronous — Includes an asynchronous reset signal for all sequential elements in the EDT logic and a new top-level pin named "edt_reset" to control it. You can rename this pin or share it with a functional pin in the design core using the set_edt_pins command.

Note

You should define the reset as a clock primary input using the add_clocks command during logic creation. This accounts for the signal off state (high or low) when writing out the EDT logic. If the signal is undefined, an active high is assumed at the boundary of the EDT logic. If the signal is shared with a global reset that is active low, you must manually insert an inverter for the reset signal before it arrives at the EDT logic boundary.

Hybrid TK/LBIST Flow Only: -reset_signal asynchronous is *mandatory*.

- -COMpactor_type {**XPress** | **BAsic**}

  Optional switch and literal pair that determines which compactor is used in the EDT logic. Options include:

  XPress — Xpress compactor. The Xpress compactor increases compression, especially for designs that generate unknown (X) values. Default setting.

  BAsic — Basic compactor. The basic compactor is the first generation compactor.

  For more information, see "Understanding Compactor Options" in the *Tessent TestKompress User's Manual*.

- -PULse_EDt_Before_shift_clocks {**ON** | **OFf**}

  Optional switch and literal pair that determines whether the EDT logic clock is pulsed before the scan chain clock. By default, the EDT logic and scan chain clock are pulsed simultaneously. For more information, see the "Pulse EDT Clock Before Scan Shift Clocks" section in the *Tessent TestKompress User's Manual*.

- -FOrce_1hot_masking {**OFf** | **ON**}

  Optional switch and literal pair that determines whether Tessent TestKompress generates test patterns that observe a single scan chain or multiple scan chains on each channel of the Xpress compactor. Options include:

  OFf — Depending on unknown states and targeted faults, Tessent TestKompress determines which scan chains to observe. Default behavior.

  ON — One scan chain is observed on each compactor channel. This setting should be used for debug purposes. Generating test patterns with this setting increases test pattern count.

This switch is only valid when the Xpress compactor is used. Observing only one scan chain output per channel allows you to easily map a failure on a channel to a specific scan cell. For more information on the Xpress compactor, see "Understanding Compactor Options" in the *Tessent TestKompress User's Manual*.

- -SEParate_control_data_channels on| off

  An optional switch that when enabled (set to on) specifies that the tool automatically determines the number of control channels and inserts control bits behind the control channels only. This is needed for channel sharing between non-identical blocks. For more information, see "EDT IP Creation With Separate Control and Data Input Channels" *Tessent TestKompress User's Manual*.

  You apply this setting per EDT block. For a design with multiple EDT blocks, you can enable some of the blocks using "-separate_control_data_channels on" and disable the remaining blocks using "-separate_control_data_channels off".

Note

You can only use this option if the user-specified number of input channels is greater than or equal to one plus the number of control channels; otherwise, the tool issues an error and aborts.

- -RETIme_chain_boundaries **ON | Off**

  An optional switch that specifies whether to add retiming cells (lockup cells) at scan chain inputs and outputs. These cells are added to ensure that each scan chain starts with an LE register and ends with a TE register. This switch replaces bypass lockup cells with retiming lockup cells that are part of the EDT scan chains; this eliminates the need for bypass-only lockup cells. For more information on bypass cells as part of the EDT logic, see "Differences Based on Inclusion/Exclusion of Bypass Lockup Cells in EDT Chains" in the *Tessent Shell Reference Manual*.

  Retiming cells are added only when lockup cells are inserted with the "-lockup_cells ON" option. These cells are included as part of both EDT and EDT-bypass scan chains.

  ON — Specifies to include retiming cells in EDT chains. This is the default for the Hybrid TK/LBIST flow (dft -edt -logic_bist context).

  Off — Specifies to *not* include retiming cells in EDT chains. This is the default for TestKompress EDT IP creation(dft -edt context).

  For complete information on including retiming cells in EDT chains, see section "Lockup Cell Analysis For Bypass Lockup Cells Included as Part of the EDT Chain" in the *Tessent TestKompress User's Manual*.

- **-lbist_misr_input_ratio *integer*** (**Hybrid TK/LBIST Flow IP Creation Only**)

  A required switch and integer pair that specifies the ratio of the number of scan chains over number of MISR inputs. The specified value should be a power of 2.

  If the value is 1, then all scan chain outputs are connected to the MISR.

If the value is greater than or equal to #scan_chains/# channels, then all EDT channel outputs are connected to the MISR.

If any value in between is used, then taps are taken from the middle of EDT compactor XOR tree corresponding to the value specified.

This argument only specifies the number of MISR inputs; the size of the MISR is implicitly determined based on this input. The minimum size of the MISR is at least 24 bits.

If this switch is not specified, a default MISR of 24 bits is used. When the design has more than 24 output channels, the default MISR size is increased to be greater than or equal to the number of output channels.

- -decompressor_seed ***hex_string*** (**Hybrid TK/LBIST Flow Fault Simulation Only**)

  An optional switch and hexadecimal pair that specifies the initial seed of the PRPG.

  The LogicBIST fault simulation dofile includes this switch with the hardware default value for arguments.

  If you intend to perform fault simulation with a different starting PRPG seed, then modify the ***hex_string*** argument to specify the desired seed.

Example 1

The following example sets up a EDT logic for a design with four scan chains. The EDT logic is set up for one scan channel and no bypass circuitry.

```
add_scan_groups grp1 atpg.proc
add_scan_chains chain1 grp1 edt_si1 edt_so1
add_scan_chains chain2 grp1 edt_si2 edt_so2
add_scan_chains chain3 grp1 edt_si3 edt_so3
add_scan_chains chain4 grp1 edt_si4 edt_so4
add_clocks 0 clk
set_edt_options -channels 1 -bypass_logic off
set_system_mode analysis
```

Example 2

The following example generates EDT logic with two input channels and one output channel. The number of bypass chains defaults to 1, corresponding to the smaller of the two specified channel values.

```
set_edt_options -input_channels 2 -output_channels 1
```

Example 3 (Hybrid TK/LBIST Flow IP Creation)

This command during IP creation is used to connect the scan chains to MISR by combining four scan chain outputs using XOR gates to each MISR input.

```
set_edt_options -lbist_misr_input_ratio 4
```

Example 4 (Hybrid TK/LBIST Flow Fault Simulation)

The following example shows how to modify the PRPG seed in analysis mode during fault simulation:

```
set_edt_options -decompressor 00000001
```

```
set_system_mode analysis
simulate_patterns -source bist -store_patterns all
set_edt_options -decompressor 00110101
simulate_patterns -source bist -store_patterns all
```

Related Commands

report_edt_configurations
report_edt_pins
report_scan_volume
set_bypass_chains
set_compactor_connections
set_edt_pins
write_edt_files

set_edt_pins

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Usage

Specifying a new or shared EDT pin

set_edt_pins {{Input_channel | Output_channel} {index [*pin_name*] [*internal_node_name*]} |
{*edt_pin* [*pin_name*] [*internal_node_name*]} |
{***pin_name*** -ON_edt_module}}

Renaming and/or inverting EDT pins

set_edt_pins {{Input_channel | Output_channel} {{index [*pin_name*]} | {*edt_pin*}}
[-Noinv | -Inv]}

Inserting channel pipeline stages

set_edt_pins {{Input_channel | Output_channel} {[*index*]{-PIpeline_stages ***#stages***}}}

Specifying the change edge for compactor outputs

set_edt_pins [Output_channel -CHange_edge_at_compactor_output
{Any | LEading_edge_of_edt_clock | TRailing_edge_of_edt_clock}]
| [Output_channel *index* -CHange_edge_at_compactor_output
{Any | TRailing_edge_of_edt_clock}]

Specifying internally driven EDT control pins

set_edt_pins {{*edt_pin*} {-} [*internal_node_name*]}

Description

By default, all EDT pins are created as dedicated pins with default names and no inversions. The same set_edt_pins command must be used for both the pattern generation phase and the EDT logic creation phase to generate patterns correctly for modified pins.

Except for added channel pipeline stages, a dofile generated during the EDT logic creation phase contains the correct set_edt_options commands for the pattern generation phase. If you add channel pipeline stages, you must specify those changes in the pattern generation phase with another set_edt_pins command.

Use the set_edt_pins command for the following operations:

- **Specify a new or shared EDT pin**

  When an EDT channel output pin is shared with a functional output pin, a multiplexer is inserted in the top-level wrapper to share the signals between the output pins. The multiplexer is controlled by the scan_enable signal. If you rename an EDT pin with a name already used by a functional pin in the core design, the EDT pin shares the functional pin.

- **Rename an EDT control pin**

To rename an EDT pin, specify the *edt_pin* along with a *pin_name* argument that specifies the new name. To rename an EDT channel, specify an input/output channel and index number for the channel along with a *pin_name* argument that specifies the new name. To apply the new name to the pin at the EDT module level in the RTL, use the *pin_name* -On_edt_module switch.

- **Specify an inverted signal between the chip pin and the EDT logic**

  I/O pads added outside the EDT logic sometimes invert their signal. Use the *-Inv* switch to specify the inversion of the EDT control pins during EDT logic creation. This ensures that generated test procedures and dofiles operate correctly in the pattern generation phase after the I/O pads are synthesized. The *pin_name* argument is optional if you are only changing the inversion status of an EDT pin.

- **Specify that an EDT control pin is driven internally**

  For more information, see "Internally Driven EDT Pins" in the *Tessent TestKompress User's Manual*.

- **Specify channel pipeline stages**

  Channel pipeline stages are pipeline stages added outside the EDT logic between a top level channel input pin/pad and the decompressor input or compactor output.

  Channel pipeline stages are typically added during top-level design integration to reduce signal propagation time between the top-level pads and the channel inputs and outputs. Channel pipeline stages are not created as part of the EDT logic, but they must be specified to generate the correct test patterns.

  To specify channel pipeline stages for a particular channel during the pattern generation phase, use the *index* number of the channel, the *-Pipeline_stages* switch, and the number of pipeline stages.

- **Specify the change edge for compactor outputs**

  You can use this feature to specify whether compactor output data changes on the LE time or whether the tool aligns the change edge at compactor output with the TE of the EDT clock.

  By default, the compactor output data changes on the TE time of the EDT clock. There are two usage scenarios as follows:

  Without compactor pipelining (compactor pipeline register has not been inserted)

  - If LE is specified for all channels, a pipeline stage clocked by edt_clock (LE) is added at the compactor output; the pipeline stage is only added if any of the last scan cells change at any time other than the LE of the EDT clock. Lockup cells are added if necessary either before the pipeline stage or at the output of the scan chains on a per-compactor basis.

- If TE is specified for a channel, lockup cells driven by the TE of the EDT clock are added to all channels when necessary to ensure a TE change edge.

With compactor pipelining (compactor pipeline register has been inserted)

- The compactor pipeline guarantees the compactor output changes on the LE of EDT clock because the pipeline stage inside the compactor is driven by edt_clock (LE). If the number of levels of logic is less than or equal to the number of pipeline levels specified with set_edt_options, a pipeline stage is added only when necessary to ensure an LE change. In addition, lockup cells are added when needed.
- If TE is specified for a channel, an additional lockup cell is added on the output of every compactor to ensure the TE change edge.

The dofile written out during EDT logic creation reflects the pipeline stages added to meet the speed objectives or to ensure that the output of the compactor changes at the positive edge of the EDT clock.

For more information, see "EDT Control and Channel Pins" and "Using Pipeline Stages Between Pads and Channel Inputs or Outputs" in the *Tessent TestKompress User's Manual*.

Arguments

- Input_channel

  Optional literal that specifies an EDT input channel pin.

- Output_channel

  Optional literal that specifies an EDT output channel pin.

- *index*

  Optional integer that specifies the index number for a specified channel pin. The index must be positive and have a valid range from one up to the total number of scan channels.

- *pin_name*

  Optional replaceable string that specifies a pin name. Depending on usage, the *pin_name* may reflect the name of an existing functional pin in the core design or EDT logic, an existing pin on the EDT module, or the new name for an existing pin. This is a required argument if you are using the internal_node_name argument to create a connection between this pin and an internal node.

- *internal_node_name*

  Optional replaceable string that specifies the hierarchical name of an internal node to connect an EDT pin to. This argument defines the connection between an EDT pin and the internal node of an I/O pad for internally placed EDT logic.

  When defining the connection, you must also include a *pin_name* argument in the same command. If only one string is specified, the tool interprets it as the *pin_name*.

- *edt_pin*

  Optional literal that specifies an EDT control pin. Literal options include:

  Clock — edt_clock pin.

  Reset — edt_reset pin.

  Scan_enable — top-level scan_en pin.

  Bypass — edt_bypass pin.

  Single_bypass_chain — edt_single_bypass_chain pin.

  Update — edt_update pin.

  COnfiguration — edt_configuration pin for dual configuration applications.

  Master_clock — master_clock pin. Use in latch-based EDT logic for LSSD designs only.

  Slave_clock — slave_clock pin. Use in latch-based EDT logic for LSSD designs only.

  Low_power_shift_en — low power shift enable pin for low-power decompressor.

- **[**-Noinv | -Inv]

  Optional switch that specifies an EDT pin signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.

  You should specify -inv in the following situations:

  - **External flow** — Inversion between the wrapper created by the tool and the chip boundary.
  - **Internal flow** — Inversion between the internal connection node for the EDT pin and the chip boundary.

  Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.

- -ON_edt_module

  Optional switch that specifies a pin on the EDT module in the EDT RTL. Use this argument to apply a change to a specified pin at the EDT module level only.

- -Pipeline_stages #stages (Pattern Generation Phase only)

  Optional switch and non-negative integer pair that specifies the number of pipeline stages on specified input or output channels. Use the channel *index* argument to apply a number of channel pipeline stages to a specific channel. If no *index* argument is used, the specified number of channel pipeline stages applies to all input or all output channels for the current EDT block.

  Use this argument during the pattern generation phase to specify the number of pipeline stages inserted between top-level pins or pads and EDT channel inputs or outputs.

If your design contains channel pipeline stages, you must also make some changes in the test procedure file to accommodate them.

Note

When EDT Finder is enabled, the tool updates the number of pipeline stages with what was traced even if you have specified a different number using the -pipeline switch.

Note

You must generate test patterns for existing bypass scan chains from scratch. You cannot create test patterns for existing bypass chains from identical test patterns created for EDT. For more information on using bypass chains defined in the core logic, see "Generating EDT logic When Bypass Logic is Defined in the Netlist" in the *Tessent TestKompress User's Manual*.

For more information, see "Using Pipeline Stages Between Pads and Channel Inputs or Outputs" in the *Tessent TestKompress User's Manual*.

- [Output_channel -CHange_edge_at_compactor_output {Any | LEading_edge_of_edt_clock | TRailing_edge_of_edt_clock}] | [Output_channel *index* -CHange_edge_at_compactor_output {Any | TRailing_edge_of_edt_clock}]

  Optional switch and literal pair that specifies which edge the compactor output data changes on. This switch does not support latch-based EDT. Options include:

  Any — Specifies that data can change at either edge and no additional logic is required.

  TRailing_edge_of_edt_clock — Specifies that compactor outputs need to change on the TE edge of the EDT clock. You can use the *index* option to specify a change edge for individual channels because it does not introduce an additional shift.

  LEading_edge_of_edt_clock — Specifies that all compactor outputs need to change on the LE edge of the EDT clock. An *index* cannot be specified because it may introduce varying amounts of shift cycles across output channels. When you specify this option and all scan chain outputs already change at LE, the tool adds a lockup cell and LE compactor pipeline register.

- -edt_pin

  Optional switch that specifies there is no top-level pin associated with an *edt_pin* that is driven internally. Use this switch during EDT logic creation and test pattern generation for the internal flow and during test pattern generation for the external flow to prevent false K5 DRC violations.

Note

Input and output channels must always have a corresponding top-level pin.

For more information, see "Internally Driven EDT Pins" in the *Tessent TestKompress User's Manual*.

Example 1

The following example configures the EDT logic with two scan channels, reports the names and inversion status of the EDT pins, and issues four set_edt_pins commands that:

1. Share the EDT clock pin with a functional pin, a1.
2. Rename the dedicated EDT bypass pin to my_bypass and specifies the bypass signal is inverted between the PI pin and the EDT logic.
3. Share EDT channel 1 input pin with functional pin, portain[7].
4. Change the inversion status of the edt_update pin.

A final report_edt_pins command displays the new EDT pin settings.

set_edt_options -channels 2
report_edt_pins

```
//
//    Pin description          Pin name           Inversion
//    ---------------          --------           ---------
//    Clock                    edt_clock              -
//    Update                   edt_update             -
//    Bypass mode              edt_bypass             -
//    Scan channel 1 input     edt_channels_in1       -
//      "     "    " output    edt_channels_out1      -
//    Scan channel 2 input     edt_channels_in2       -
//      "     "    " output    edt_channels_out2      -
//
```

set_edt_pins clock a1
set_edt_pins bypass my_bypass -inv
set_edt_pins input_channel 1 portain[7]
set_edt_pins update -inv
report_edt_pins

```
//
//    Pin description          Pin name           Inversion
//    ---------------          --------           ---------
//    Clock                    a1                     -
//    Update                   edt_update            inv
//    Bypass mode              my_bypass             inv
//    Scan channel 1 input     portain[7]             -
//      "     "    " output    edt_channels_out1      -
//    Scan channel 2 input     edt_channels_in2       -
//      "     "    " output    edt_channels_out2      -
//
```

Assume the settings defined by the preceding example are specified by similar set_edt_pins commands during the pattern generation phase. The next example specifies there are three pipeline stages between the scan channel 1 input pin (portain[7]) and the corresponding top level pad, and changes the inversion status of the pin. The example then displays the updated settings. (The Pin description column is not shown in this example in order to show the reporting command's output when pipeline stages are defined.)

set_edt_pins input_channel 1 -inv -pipeline_stages 3
report_edt_pins

```
//
//  Pin name            Inversion       Channel pipeline stages
//  --------            ---------       -----------------------
//  a1                      -
//  edt_update             inv
//  my_bypass              inv
//  portain[7]             inv                   3
//  edt_channels_out        -                    0
//  edt_channels_in2        -                    0
//  edt_channels_out2       -                    0
```

The next example specifies new pin names to be used at the “edt” module interface for the two EDT input channels, the two EDT output channels, and the EDT clock signal:

```
//Input ch: use the name, ch_in_bus, at the edt-IP RTL module interface.
set_edt_pins input_channel ch_in_bus -on_edt_module

//Output ch: use the name, ch_out_bus, at the edt-IP RTL module interface.
set_edt_pins output_channel ch_out_bus -on_edt_module

//EDT clock: use the name tk_clk_P at the edt-IP RTL module interface.
set_edt_pins clock tk_clk_P -on_edt_module
```

The next example shows a complete dofile for an external logic location flow that implements user-specified EDT pin names at the “edt” module level:

```
dofile scan_setup.dofile

set_edt_options -channels 2 -bypass off

//
//clock: design level=PSEI, edt module level=tk_clk_P
set_edt_pins clock PSEI
set_edt_pins clock tk_clk_P -on_edt_module

//
//channel inputs: design level = AI[1]/AI[2], edt module level=ch_in_bus
set_edt_pins input_channel 1 AI[1]
set_edt_pins input_channel 2 AI[2]
set_edt_pins input_channel ch_in_bus -on_edt_module

//
//channel outputs: design level=OA[1]/OA[2], edt module level=ch_out_bus
set_edt_pins output_channel 1 OA[1]
set_edt_pins output_channel 2 OA[2]
set_edt_pins output_channel ch_out_bus -on_edt_module

set_system_mode analysis
report_edt_pins

write_edt_files created -verilog -replace
```

The next example shows “edt” and “edt_top” modules (external logic location flow), with RTL changes resulting from the use of the -On_edt_module switch highlighted in bold. For brevity, only the relevant portions of the “edt_top” module are shown.

```
/*********************************************************************
** Module: my_core_edt
**
*********************************************************************/
module my_core_edt (tk_clk_P, edt_update, ch_in_bus, ch_out_bus,
                    edt_scan_in, edt_scan_out);
   input           tk_clk_P;
   input           edt_update;
   input    [ 1:0] ch_in_bus;
   output   [ 1:0] ch_out_bus;
   output   [15:0] edt_scan_in;
   input    [15:0] edt_scan_out;

   wire edt_mask;

   my_core_edt_decompressor my_core_edt_decompressor_i
                                          (.edt_clock(tk_clk_P),
                                           .edt_update(edt_update),
                                           .edt_channels_in(ch_in_bus),
                                           .edt_mask(edt_mask),
                                           .edt_scan_in(edt_scan_in));

   my_core_edt_compactor my_core_edt_compactor_i
                                          (.edt_clock(tk_clk_P),
                                           .edt_update(edt_update),
                                           .edt_scan_out(edt_scan_out),
                                           .edt_mask(edt_mask),
                                           .edt_channels_out(ch_out_bus));

endmodule


/*********************************************************************
** Module: my_core_edt_top
**
*********************************************************************/
module my_core_edt_top (NMOE,
                        NMWE,
                        DLM,
                        ...
                        scan_en,
                        edt_update);

   output NMOE;
   output NMWE;
   output DLM;
   ...
   input    scan_en;
   input    edt_update;
   ...
   my_core my_core_i (.NMOE(NMOE),
                      .NMWE(NMWE),
                      .DLM(DLM),
```

```
                    ...
                    .scan_en(scan_en));

    my_core_edt my_core_edt_i (.tk_clk_P(PSEI),
                        .edt_update(edt_update),
                        .ch_in_bus({AI[2], AI[1]}),
                        .ch_out_bus({edt_channels_out_2,
                                     edt_channels_out_1}),
                        .edt_scan_in(edt_scan_in),
                        .edt_scan_out(edt_scan_out));
    ...
endmodule
```

Example 2

The following example specifies that the output of the compactor should change at the LE of the clock.

set_edt_pins output -change_output leading

If the compactor has any outputs that change data on TE, a pipeline stage and the necessary lockup cells are added at the output of every compactor.

Example 3

The following example uses a design with 100 scan chains and the following parameters for compression:

set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 2
set_edt_pins output -change_output leading

Pipeline stages are added after the second XOR gates stage in the compactor. Accordingly, lockup cells are also added. However, no other pipeline stages are added at the output of the compactor because the compactor pipelining guarantees that the output of the compactors will change with respect to the LE of the clock.

Assume the same design and change the pipeline stages to:

set_edt_options -channels 10 -pipeline_logic_levels_in_compactor 6
set_edt_pins output -change_output leading

Now, no compactor pipeline stages are needed due to the minimum logic levels allowed in the compactor by the user. Under such circumstances, the last scan cell of all chains in the design are analyzed and if any of the scan outputs change at a TE edge, then the tool adds a pipeline stage at the output of all the compactors along with necessary lockup cells.

Example 4

The following example shows the commands used for a design with 100 chains and 10 channel inputs/outputs with some channel pins with pipelining.

set_edt_options -channels 10
set_edt_pins output 1 -change_edge trailing
set_edt_pins output 5 -change_edge trailing

Lockup cells are added in the compactors that have pipeline stages in the corresponding channels.

Example 5

The following example sets the output of all compactors to change at LE except for one that changes on TE because of channel output pipelining.

```
set_edt_options -channels 10
set_edt_pins output -change_edge leading
set_edt_pins output 5 -change_edge trailing
```

This guarantees that all compactor outputs will change at LE except for channel 5, which would change at TE with respect to the EDT clock.

Example 6

The following example specifies the EDT update control pin is internally driven by the */top/xyz/out* node for Tessent TestKompress logic created with the internal flow.

```
set_edt_options -location internal
set_edt_pins update - /top/xyz/out
```

The following example specifies the EDT update control pin has no corresponding top-level pin during test pattern generation.

```
set_edt_pins update -
```

Example 7

The following example specifies the connections between the channel input and output pins and their internal connection points for an internal flow. Notice that the top-level pins port_x and port_y are specified because the internal_node_name argument is used.

```
set_edt_options -location internal
set_edt_pins input_channel 1 port_x /ip_pad_block/port_x_pad/Z
set_edt_pins output_channel 1 port_y /ip_pad_block/port_y_pad/A
```

Related Commands

report_edt_configurations
report_edt_pins
set_edt_options
write_edt_files

set_edt_power_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_edt_power_controller **Shift** {None | Enabled | Disabled}
[-MIN_Switching_threshold_percentage *percentage_integer*]

Description

Generates and inserts a power controller in the compression logic to enable low-power shift.

This command controls the bias of the low-power decompressor in the EDT IP creation phase. For more information, see "Low-Power Shift and Switching Thresholds" in the *Tessent TestKompress User's Manual*.

In the IP Creation phase, you can use this command to create the power controller, to specify its hardware capability (switching threshold), and to configure the generated dofile and test procedure file.

In the Pattern Generation phase, you can only use this command to indicate that the controller is present and whether it is enabled in the test procedure file read by the tool.

The power controller may be configured in both phases as either "enabled" or "disabled". If you are using the same configuration (enabled/disabled) during both IP creation and pattern generation, then this command is already included in the generated dofile and you do not need to make any changes to the controller configuration. However, if you want to change how it was configured during IP creation, you need to specify the opposite value (enabled/disabled).

To generate patterns with the low power controller hardware enabled, you must do the following:

1. In the test procedure file, ensure the edt_low_power_shift_en is forced to the correct values during the shift procedure. For more information on the edt_low_power_shift_en signal, see "Power Controller Logic" in the *Tessent TestKompress User's Manual*.
2. Issue a "set_edt_power_controller Shift Enabled" command in Setup mode to enable the power controller before you generate test patterns.

To generate patterns with the low power controller hardware disabled, you must do the following:

1. In the test procedure file, ensure the edt_low_power_shift_en is forced to the correct values during the shift procedure. For more information on the edt_low_power_shift_en signal, see "Power Controller Logic" in the *Tessent TestKompress User's Manual*.
2. Issue a "set_edt_power_controller Shift Disabled" command in Setup mode to disable the power controller before you generate test patterns.

Note

You cannot use this command during ATPG to change the switching threshold of an already created controller; instead, use the set_power_control command to change the actual switching percentage during ATPG.

For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

Arguments

- **Shift**

  Required literal that specifies the generated power controller should enable low-power shift.

- None

  Required literal that disables creation of the power controller in the compression logic. This is the default setting.

- **Enabled**

  Required literal that during IP creation creates the power controller hardware as part of the compression logic and writes the ATPG dofile that enables the hardware and a test procedure file that forces the edt_low_power_shift_en control signal to its enabled value.

  During ATPG, this option tells the tool that the low power controller is enabled in the test procedure file read by the tool.

- **Disabled**

  Required literal that during IP creation creates the power controller hardware as part of the compression logic and writes the ATPG dofile that disables the hardware and a test procedure file that forces the edt_low_power_shift_en control signal to its disabled value.

  During ATPG, this option tells the tool that the low power controller is disabled in the test procedure file read by the tool.

- -MIN_Switching_threshold_percentage *percentage_integer*

  Optional switch and integer that specifies the minimum switching threshold value used to determine power controller parameters, such as input shift register size. Valid integer options are 1 to 50. If an integer greater than 50 is specified, the tool uses 50. The default setting is 15%.

  The tool uses this value to calculate the size of the power controller. The power controller must be large enough to control the number of scan chains required to achieve the minimum switching threshold specified. For more information, see "Low-Power Shift and Switching Thresholds" in the *Tessent TestKompress User's Manual*.

Example 1

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold, and creates the dofile and test procedure file that enable the hardware during pattern generation.

```
set_edt_power_controller shift enabled -min_switching_threshold_percentage 20
…
set_system_mode analysis
write_edt_files low_power_enabled
```

Given the compression logic created with the preceding command, use the following commands during ATPG phase to turn the power controller on with a 22% switching threshold for scan chain loading and to generate test patterns.

```
set_power_control shift –switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold, and creates the dofile and test procedure file that disable the hardware during pattern generation.

```
set_edt_power_controller shift disabled -min_switching_threshold_percentage 20
…
set_system_mode analysis
write_edt_files low_power_disabled
```

Given the compression logic created with the preceding command, use the following commands to enable the power controller and set it to a 22% switching threshold. This application also requires that you modify the edt_low_power_shift_en pin value in the test procedure file.

```
set_edt_power_controller shift enabled
set_power_control shift –switching_threshold_percentage 22
set_system_mode analysis
...
create_patterns
```

Related Commands

report_edt_configurations
report_power_metrics
set_power_control

set_external_capture_options

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

Mode: setup (patterns -scan context only), analysis

Usage

set_external_capture_options { OFf | -MINimum_cycles *cycles* | -FIXed_cycles *cycles* | { -PLL_cycles *cycles timeplate_name* }} | { -CAPture_procedure *procedure_name* }

Description

Specifies the number of tester cycles to be used in the capture window of all patterns.

The set_external_capture_options command allows you to specify a minimum or fixed number of external tester capture cycles when saving -mode_external patterns for the current test session, including chain test patterns with capture cycle and retention test patterns. The integer you supply with this command specifies the minimum or fixed number of tester cycles that must be present between the end of the load_unload procedure and the start of the next pattern, or next load_unload procedure for multi-load patterns. When saving –mode_internal patterns for purposes other than pattern retargeting, the tool ignores set_external_capture_options settings and saves the patterns created by the ATPG engine, using the clock_sequential and capture procedures as is normally done.

Note

The options you specify with the set_external_capture_options command apply to all pattern retargeting patterns, including mode_internal.

The integer you supply, for fixed or minimum, adds extra padding cycles to the end of the capture window if the number of tester cycles used in the pattern is less than the specified number when saving mode-external simulation or tester interface format patterns. The ASCII and binary format patterns are not affected by these padding cycles. The number of tester cycles you designate also applies to scan loads used in multi-load patterns, so the launch cycles used between scan loads in the multi-load pattern is also padded to ensure the same number of tester cycles between scan loads.

The padding cycle is created from a copy of the last cycle from the capture procedure with all clocks turned off, all measure data removed, and all free running clocks turned on. Clocks that are marked as always_capture, as well as free-running clocks, are pulsed in these padding cycles. This padding cycle is added after the last cycle of the capture procedure and before the first cycle of the next pattern.

There is also a -PLL_cycles switch that is meant for use with clock_control definitions only. To use the switch, all clocks must either be controlled by clock_control definitions, be constrained, or be free running or always capture. Also, hold_pi must be on, and mask_po must be on for all outputs. The -PLL_cycles switch doesn't so much add padding cycles as it replaces the ATPG cycles with a set number of external capture cycles using the specified timeplate. It is somewhat

similar to internal_mode and external_mode of the named capture procedures. The internal_mode is what is generated by the ATPG engine using the above restrictions, and the external_mode is the number of cycles specified with the designated timeplate, and the clocks that pulse are the free running or always-capture clocks. It is possible to specify the number of pll_cycles to be less than the max_sequential depth setting. As with the -MINimum_cycles or -FIXed_cycles, -PLL_cycles is only used when saving -Mode_external patterns.

When using -PLL_cycles, instead of an annotation that lists the number of padding cycles being added, there is an annotation that lists the original ATPG sequential depth for this pattern.

When STIL and WGL patterns are read back into the ATPG tools, the tool flags padding cycles added to the end of the capture window as added dummy cycles and does not use those padding cycles when converting the test cycles into ATPG pattern data. However, the tool uses these cycles when converting cycle-based fail information into pattern-based fail data.

As an alternative to directly specifying the number of external tester capture cycles, you can use the -CAPture_procedure switch and string pair to specify an external capture procedure that defines the external capture cycles to use with clock control definitions.

Issuing the command with no options reports the current settings. Any subsequent use of this command overrides all previous settings.

Arguments

- OFf

  An optional literal that turns off any previous tester cycle settings. Note, however, that this option does not automatically disable maximum sequential depth. You have to issue set_pattern_type –Max_sequential Off to disable the setting for the maximum sequential depth.

- -MINimum_cycles *cycles*

  An optional switch and integer pair that specifies the minimum number of cycles present in the capture window when saving patterns in STIL or WGL format.

- -FIXed_cycles *cycles*

  An optional switch and integer pair that specifies the fixed number of tester cycles that needs to be present in the capture window when saving patterns in STIL or WGL format. The tool automatically calls “set_pattern_type –Max_sequential” to set the maximum sequential depth to be *integer* if the setting for maximum sequential depth is disabled or the maximum sequential depth is less than *cycles*.

- -PLL_cycles *cycles timeplate_name*

  An optional switch that specifies the fixed number of PLL capture cycles (integer value) and string that specifies the name of a timeplate that was loaded with the procedure file when saving patterns in STIL or WGL format.

- -CAPture_procedure *procedure_name*

  An optional switch and string pair that specifies the name of an external_capture procedure, which is used for all capture cycles between each scan load, even when the pattern is a

multi-load pattern. For more information about the external_capture procedure, refer to the section "Capture Procedures (Optional)" in the *Tessent Shell User's Manual*.

Example

The following example adds one PLL capture cycle to the end of the capture window:

set_external_capture_options -pll_cycles 1 pll_timeplate

Related Commands

set_pattern_type

set_external_simulator

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

Mode: setup, analysis

Usage

set_external_simulator {**-SIMulation_script** ***script_filename***} | {**vsim** [***options***…]}

Description

Specifies the shell command to use to invoke an external simulator for verifying tool-produced patterns.

The command string must be valid for the simulator; therefore, you may need to refer to the simulator's user documentation for correct syntax and arguments.

Arguments

- **-SIMulation_script** ***script_filename***

  A required switch and string pair that specifies a simulation script to use for verifying patterns. This script will be used automatically for the analyze_simulation_mismatches command. Supported simulation tools include vsim, ncverilog, and vcs.

  For a script example, refer to this "Example" in the *Tessent Scan and ATPG User's Manual*.

- **vsim** [***options***…]

  A required literal and optional string that specifies a basic invocation command for Model*Sim* in batch mode.

Example 1

The following example checks the current external simulator setting, then specifies to invoke Model*Sim* in batch mode with loading messages disabled, when verifying simulation results:

report_external_simulator

```
../vsim -c
```

set_external_simulator vsim -c -quiet
report_external_simulator

```
../vsim -c -quiet
```

Model*Sim* supports simulation with Standard Delay Format (SDF) back-annotation. Generally, you can apply the timing information in an SDF file to any instance in a design.

Example 2

The following example includes arguments that direct the simulator to annotate maximum timing values from the SDF timing file *cpu1.sdf* to instance /my_design/inst1, and minimum timing values from the SDF timing file *cpu2.sdf* to instance /my_design/inst2:

```
set_external_simulator vsim -c -sdfmax /my_design/inst1=cpu1.sdf
        -sdfmin /my_design/inst2=cpu2.sdf
```

Related Commands

analyze_simulation_mismatches
report_external_simulator
report_measure_cycles

set_fails_report

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

Mode: setup, analysis

Usage

set_fails_report **OFf** | **ON**

Description

Specifies whether the design rules checker displays clock rule failures.

The set_fails_report command displays all clock rule failures of the design rules checker. The default mode upon invocation of the tool is Off.

Arguments

- **OFf**

  A literal that specifies for the design rules checker to not display clock failures. This is the default upon invocation of the tool.

- **ON**

  A literal that specifies for the design rules checker to display clock failures.

Example

The following example displays clock failures from the design rules checking process:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
add_clocks 1 clock1
add_clocks 0 clock2
set_fails_report on
set_system_mode analysis
```

Related Commands

add_clocks
delete_clocks
report_clocks

set_fault_mode

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

Mode: setup, analysis

Usage

set_fault_mode **Uncollapsed** | **Collapsed**

Description

Specifies whether the fault mode is collapsed or uncollapsed.

The **set_fault_mode** command specifies whether the tool uses collapsed or uncollapsed fault lists for fault counts, test coverages, and fault reports. The default fault mode upon invocation of the tool is Uncollapsed. When you display a report on uncollapsed faults, the tool lists the representative fault first followed by its equivalent faults.

The DFTVisualizer Browser and Library Browser windows display fault information based on this setting.

Note

This command is not valid for the bridge logic fault model (specified with the set_fault_type Bridge -Static command).

Arguments

- **Uncollapsed**

  A literal specifying that the tool include equivalent faults in the fault lists. This is the default mode upon invocation of the tool.

- **Collapsed**

  A literal specifying that the tool not include equivalent faults in the fault lists.

Example 1

The following example sets the fault mode to collapsed and then displays only the collapsed faults:

```
set_system_mode analysis
add_faults -all
set_fault_mode collapsed
report_faults -all
```

Example 2

The following shows an example when reporting uncollapsed tied faults as compared to reporting collapsed tied faults:

```
Uncollapsed:      Collapsed:
0 TI /I_140/I     0 TI /I_140/I
1 TI /II_140/O    1 TI /II_140/O
1 EQ /II_140/I
```

Related Commands

add_faults
delete_faults
read_faults
report_faults
report_testability_data
set_fault_sampling
set_fault_type
write_faults

set_fault_protection

Context: dft -edt, patterns -scan

Mode: analysis

Note

This command does not apply to bridge or UDFM faults.

Usage

set_fault_protection **ON** | **OFf**

Description

Specifies for the tool to protect detected faults from manipulation by subsequent tool commands.

The set_fault_protection command specifies for the tool to protect all detected faults currently in memory from manipulation by subsequent tool commands. The tool counts protected faults in calculations of coverage but does not try to generate patterns for them. When performing multiple ATPG runs, use this command after each one to get the tool, on the next run, to generate tests only for remaining undetected faults (stuck-at, toggle, transition, or IDDQ), yet determine coverage statistics based on the cumulative total of detected faults.

Note

To protect detected faults read in from an external fault list file, you must load them using the read_faults -Protect command.

Caution

The set_fault_type command erases all fault information, even the information about protected faults. If you turn on fault protection, be sure you no longer need the protection before issuing a subsequent set_fault_type command.

Arguments

- **ON**

  A literal that specifies to protect all detected faults currently in memory from manipulation by subsequent tool commands.

- **OFf**

  A literal that cancels the protection for previously protected faults. If the faults had been protected by a “set_fault_protection on” command, the original fault classes are preserved when you turn off protection. If the faults had been protected by a “read_faults -protect” command, formerly protected faults have the DI fault class when you turn off protection. When you remove protection from faults, tool commands apply to them the same as to faults that never had protection. This is the default.

Note

Be aware once you turn off protection, you cannot re-declare the same formerly protected faults as protected again. This is because the command applies to all detected faults in memory at the time you enter the command. Suppose, for example, you do an ATPG run, then turn on protection and do a second run. If you now turn protection off, the tool adds the formerly protected faults to the detected faults from the second run. Turning on protection at this point simply protects them all.

Example

The following example generates and saves patterns for three different capture procedures, while protecting the faults detected in any particular run from being reset by the run that follows it. Notice how the “set_fault_protection on” command is reentered after each run in order to add the newly detected faults to the tool’s internal list of protected faults.

```
set_system_mode analysis
set_capture_procedure on my_cap_proc1
create_patterns
write_patterns my_cap_proc1_pat
set_fault_protection on

set_capture_procedure on my_cap_proc2
create_patterns
write_patterns my_cap_proc2_pat
set_fault_protection on

set_capture_procedure on my_cap_proc3
create_patterns
write_patterns my_cap_proc3_pat
set_fault_protection on
```

Related Commands

read_faults
set_capture_procedures

set_fault_sampling

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

Mode: setup, analysis

Usage

set_fault_sampling {**Off** | ***percentage***} [-Seed *integer*]

Description

Specifies the fault sampling percentage used for circuit evaluation or scan identification.

Fault sampling enables you to process a fraction of the total faults to decrease processing time for evaluating large circuits. For faults other than bridge faults, once you specify a percentage, the tool randomly picks the fault samples to process. For bridge faults, the tool randomly picks bridges to process; if a particular bridge is included in a bridge sample, all faults associated with that bridge are included.

Caution

Be aware when using Tessent FastScan or Tessent TestKompress that fault sampling erases protected fault information. If some faults are protected, be sure you no longer need the protection before you issue this command.

Arguments

- **Off**

  A switch that turns off fault sampling. This is the invocation default.

- ***percentage***

  A required, positive number that specifies the fault sampling percentage used for circuit evaluation or scan identification. Any number greater than 0 and less than or equal to 100 can be specified. The default is 100.

- -Seed *integer*

  A switch and hex number pair that specifies a seed value to use in the selection of fault samples. Specifying unique seed values for different runs can provide more accurate fault sampling results. The number value must be a lower case, 32-bit hex representation. 0 is not a valid seed value. The default is 0xcccccccc.

Example 1

The following example performs a trial ATPG run in Tessent FastScan using a fault sampling of one percent of the total faults. Because the set_fault_sampling command overwrites parts of the original fault list, after the test run the example restores the fault list to the state it was in prior to the set_fault_sampling command:

```
set_system_mode analysis
add_faults -all
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
add_faults -all
```

Example 2

The following example performs the trial ATPG run, as in the preceding example, but uses faults from an external fault file named *my_fault_file*. The example then recovers from the set_fault_sampling command by clearing the internal fault list and reloading faults from the external file:

```
set_system_mode analysis
read_faults my_fault_file
set_fault_sampling 1
create_patterns
delete_faults -all
set_fault_sampling 100
read_faults my_fault_file
```

Related Commands

add_faults

report_faults

set_fault_mode

set_fault_protection

set_fault_type

write_faults

set_fault_subclass_analysis

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

Mode: setup, analysis

Usage

set_fault_subclass_analysis {-Enable | -Disable} {*fault_class*[.*fault_subclass*] … }

Description

Turns on or off the analysis for the specified fault class(es).

The set_fault_subclass_analysis command supports all of the fault classes and sub-classes supported by the set_relevant_coverage command as well as the DI.EDT fault sub-class. However, to enable the analysis of a class or sub-class of faults, you must issue this command *before* the faults have been added.

By default, all AU sub-class analyses are enabled, and all UU sub-class analyses are disabled. Note, AU.EDT and AU.FP sub-classes are enabled by default *only if* the tool can identify the EDT-related faults and false path faults. The tool can identify EDT-related faults if the EDT logic was generated with the default name or if the user specified the naming of the EDT module. The tool can identify false path faults if they were defined using the add_false_paths command or if the SDC file was read into the tool. The UU sub-class analysis, when enabled, is performed only for stuck, transition, and bridge fault types.

By default, the DI.EDT fault sub-class is enabled if EDT Finder is ON. You can override this default and disable DI.EDT analysis using this set_fault_subclass_analysis command. The DI.EDT fault sub-class cannot be enabled if EDT Finder is OFF. For more information, see "Fault Sub-classes" in the *Tessent Scan and ATPG User's Manual*. For more information on the EDT Finder, see the set_edt_finder command.

You can use the set_relevant_coverage command to change the AU and UU sub-classes to include in or exclude from relevant test coverage reported by the tool.

Arguments

- -Enable

  Switch that enables analysis for the specified fault class or sub-class.

- -Disable

  Switch that disables analysis for the specified fault class or sub-class.

- {*fault_class*[.*fault_subclass*] … }

  Repeatable string that specifies the fault class or sub-class. Table 7-49 on page 1921 lists the fault classes that can be specified with this command. Table 7-50 on page 1921 list the available fault sub-classes. You can specify a fault class or sub-class either by name or code.

Example 1

The following example shows the report_statistics output when UU faults are classified into two sub-classes. Note, the set_fault_subclass_analysis command is issued first to enable UU analysis prior to adding faults.

set_fault_subclass_analysis -enable UU
add_faults -all
report_statistics

```
                              Statistics Report
                               Stuck-at Faults
-------------------------------------------------------------------------
Fault Classes                                                   #faults
                                                                (total)
-------------------------------------------------------------  -------------
  FU (full)                                                     9176206
  -------------------------------------------------------------  -------------
  UO (unobserved)                                                2138 ( 0.02%)
  DS (det_simulation)                                          7908664 (86.19%)
  DI (det_implication)                                         1172377 (12.78%)
  PT (posdet_testable)                                             226 ( 0.00%)
  UU (unused)
    CON (connected)                                                11 ( 0.00%)
    FL (floating)                                                1261 ( 0.01%)
  TI (tied)                                                      16071 ( 0.18%)
  BL (blocked)                                                   27994 ( 0.31%)
  RE (redundant)                                                 26176 ( 0.29%)
  AU (atpg_untestable)                                           21288 ( 0.23%)
-------------------------------------------------------------------------
Untested Faults
  -------------------------------------------------------
  AU (atpg_untestable)
    BB   (black_boxes)                                             116 ( 0.00%)
    PC*  (pin_constraints)                                       15453 ( 0.17%)
Unclassified                                                    5719 ( 0.06%)
  UC+UO
    AAB  (atpg_abort)                                              2138 ( 0.02%)
  *Use "report statistics -detailed_analysis" for details.
-------------------------------------------------------------------------
```

Example 2

The following example shows how to include UU.CON faults in relevant test coverage.

set_fault_subclass_analysis -enable UU
set_relevant_coverage -include UU.CON
set_relevant_coverage

```
Relevant Coverage Setting Report
-----------------------------------------
Fault Subclass          Relevant Coverage
----------------------- -----------------
AU.BB                          yes
AU.EDT                          no
AU.PC                          yes
AU.TC                          yes
AU.CC                          yes
```

```
AU.FP                                  no
AU.MCP                                yes
UU.FL                                  no
UU.CON                                yes
TI                                     no
BL                                     no
RE                                     no
-----------------------------------------
```

Related Commands

report_statistics

set_relevant_coverage

set_fault_type

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

Mode: setup, analysis

Usage

set_fault_type
Stuck |
{Iddq [-NO_SEQuential_inputs_blocked | -SEQuential_inputs_blocked]} |
TOggle |
{**TRansition** [-NO_Shift_launch] | [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFf}]} |
{**Path_delay** [-Mask_nonobservation_points]
[-ROBust_detection_only | -NO_Functional_detection]} |
Bridge |
{**Udfm** [-STATIC_fault] |
[-DELAY_fault [-NO_Shift_launch] | [-APPLY_HOLD_PI_attribute_to_shift_launch {ON | OFf}]]}

Description

Specifies the fault model type for ATPG.

The fault sites of all models are the input and output pins of the design cells in addition to external pins. 0 and 1 values are used to indicate the type of fault at the fault site for all fault models. Each fault model has its own separate fault collapsing according to the model's rules of equivalence.

When you change the fault type, both the current fault list and the internal pattern set are deleted.

Use the set_multiple_detection command to set the number of detections for each fault.

Caution

Changing to a different fault model type erases all fault information, even the information about protected faults. If some faults are protected, be sure you no longer need the protection before you issue this command.

For more information on the different fault models, refer to the *Tessent Scan and ATPG User's Manual*.

Arguments

- Stuck

  A literal that specifies the single stuck-at fault model. This is the default upon invocation of the tool.

- Iddq -NO_SEQuential_inputs_blocked | -SEQuential_inputs_blocked

  Optional literal and switch pair that specifies the IDDQ fault model. Optional switches that control the treatment of Iddq faults include:

  -NO_SEQuential_inputs_blocked — An optional switch that prevents *a priori* categorization of faults as BLocked, which must propagate through sequential primitives. This can allow creation of test patterns that would not otherwise be created, and can improve defect coverage. The additional tests created can improve defect coverage, especially for Iddq detectable defects in scan cells. This is the default.

  -SEQuential_inputs_blocked — An optional switch that retains historical behavior of *a priori* categorization of faults as BLocked which must be observed by propagation into or through a sequential primitive, such as a SL in a scan cell.

- TOggle

  A literal that specifies the toggle fault model.

- TRansition

  A literal that specifies the transition fault model.

- -NO_Shift_launch

  An optional switch that prevents the tool from creating transition tests that launch off of the last shift. Normally, for at-speed (AC) tests using transition faults, the tool tries to generate a clock sequential pattern that launches in some cycle after shift is completed (broadside). If that fails, the tool next tries to generate a basic combinational pattern that uses the last shift as the launch. You can use this switch to stop the tool from creating “launch off shift” patterns when it is unable to generate clock sequential tests. This is useful if you utilize internal PLLs and only support the first form of transition test.

Note

Pattern generation with this option requires a sequential depth of 2 if the design uses a mux-DFF scan architecture, 4 if it uses an LSSD architecture. Use the set_pattern_type command’s -Sequential switch to set the sequential depth.

Note

This option does not prevent the tool from asserting the scan enable signal during at-speed tests. You must use other means (scan_enable pin constraint or capture procedure, for example) to prevent inappropriate assertion of the scan enable pin during the at-speed part of a multiple cycle sequential transition fault test.

When you use this switch, each transition fault for which the tool fails to create a clock sequential pattern will have the status given to it by the sequential search, so you may get slightly lower fault coverage.

- -APPLY_HOLD_PI_attribute_to_shift_launch ON | OFf

  An optional switch and literal pair that determines if the hold attribute defined at primary inputs for launch-off-shift patterns is used. The default for this switch is ON.

  During test generation for launch-off-shift patterns, the tool forces primary inputs with hold attribute to have the same value as the value in the last shift cycle except for the following conditions:

  - The PI is a clock pin or a read/write control pin.
  - The PI is explicitly forced in the last shift cycle by load/unload and/or shift procedures and the forced value is not the same as the pin constraint value if there is any.
  - All scan input pins will be treated as TIE-X during test pattern generation.

  For these exception pins, the tool ignores the hold attribute defined at these pins.

- Path_delay

  A literal that specifies the path delay fault model.

- -Mask_nonobservation_points

  An optional switch that masks all non-observation points so path delay patterns are generated with values only at observation points. If there is already an internal pattern set in the tool and the fault type is already set to path delay, re-issuing the command with this switch alters the internal pattern response values by masking the non-observation points. If this switch was specified previously when setting the fault type to Path_delay, re-issuing the command without the switch causes the tool to re-simulate the internal patterns to restore the capture values at the non-observation points.

- -ROBust_detection_only

  An optional switch that generates only robust tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test, but if unable to generate a robust test, will use a non-robust test or a functional test. This switch overrides the default behavior, forcing the tool to use only robust tests. If you include this switch, the tool will classify as ATPG_untestable (AU) all path delay faults that can only be tested non-robustly or functionally. For more information about robust, non-robust and functional detection, refer to "Path Delay Fault Detection" in the *Tessent Scan and ATPG User's Manual*.

Note

This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- -NO_Functional_detection

  An optional switch that disables the generation of functional tests for path delay faults. By default, the tool first tries to detect a path delay fault using a robust test, but if unable to generate a robust test, will use a non-robust test or a functional test. This switch overrides

the default behavior, forcing the tool to use robust tests or non-robust tests, but not functional tests. If you include this switch, the tool will classify as ATPG_untestable (AU) all path delay faults that can only be tested with functional tests. For more information about robust, non-robust and functional detection, refer to "Path Delay Fault Detection" in the *Tessent Scan and ATPG User's Manual*.

Note

This switch is not cumulative; you must include it with each subsequent set_fault_type command or the tool returns to the default behavior.

- Bridge

  A literal that specifies the static bridge fault model.

- Udfm

  Optional literal that specifies a user-defined fault model (UDFM).

- -STATIC_fault

  Optional literal that enables static faults only. This is the default.

- -DELAY_fault

  Optional literal that enables delay faults only.

Example 1

The following example performs ATPG using the transition fault model:

```
set_system_mode analysis
set_fault_type transition
create_patterns
```

Example 2

The following example performs ATPG with the transition fault model, but prevents the tool from generating basic combinational patterns that launch off the last shift:

```
set_system_mode analysis
set_fault_type transition -no_shift_launch
create_patterns
```

Example 3

The next "patterns -scan" example performs ATPG with the transition fault model, specifying to target three detections per fault during pattern generation, and to detect each DS fault three times during simulation before dropping it from the fault list. Data resulting from the multiple detection setting is highlighted in bold.

Tip: When multiple detection is enabled, the number shown in the "#faults in list" column in the transcript is the cumulative tally of faults not yet detected the specified number of times. When multiple detection is not enabled, this column lists the number of faults not yet detected once. In either case, the "test cvrg" and "#faults detected" columns refer to faults detected at least once.

set_system_mode analysis
set_fault_type transition -no_shift_launch
set_multiple_detection -guaranteed_atpg_detections 3
set_pattern_type -sequential 2
create_patterns

```
//| ------------------------------------------------------------------
//|                     Analyzing the design |
//|
//|      Current clock restriction setting:      Clock_po
//|          Calling: set_clock_restriction domain_clock -any_interaction
//|
//|          Current split capture setting:      Off (optimal)
//|
//|   Current clock off simulation setting:      Off (optimal)
//|
//|            Current abort limit setting:      30
//|                               Calling:      set_abort_limit 300 100
//| -------------------------------------------------------------------
//|
//|               Current sequential depth:      0
//|               Optimal sequential depth:      2
//|    Warning: Broadside transition testing requires minimum depth of 2.
//|    Calling: set_pattern_type -sequential 2
//|
//| ---------------------------------------------------------------------
//  ATPG targeting 1 detection
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 1080
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test      BCE
//  simulated  coverage  in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---      ---       ---      ------
//   64         92.03%      772       993       61        61      84.06%
//   ---        ------      ---       ---      ---       ---      ------
//   128        96.47%      668        46       16        77      90.05%
//  ---------------------------------------------------------------------
//  ATPG targeting 2 detection, starting with 77 patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 67
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns     2-det. #faults  #faults    # eff.    # test      BCE
//  simulated  test cov. in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---      ---       ---      ------
//   192        96.47%      510        67       28       105      93.40%
```

```
//  ---------------------------------------------------------------------
//  ATPG targeting 3 detection, starting with 105 patterns
//  ---------------------------------------------------------------------
//  Simulation performed for #gates = 250  #faults = 69
//  system mode = ATPG    pattern source = internal patterns
//  ---------------------------------------------------------------------
//  #patterns     3-det. #faults  #faults    # eff.    # test      BCE
//  simulated  test cov. in list  detected   patterns  patterns    %
//  deterministic ATPG invoked with comb/seq abort limit = 300/100
//   ---        ------      ---       ---      ---       ---      ------
//   256        96.47%      409        69       25       130      94.71%
                       Statistics Report
                       Transition Faults
-------------------------------------------------------------
Fault Classes                              #faults
                                           (total)
----------------------------  --------------------------------
  FU (full)                                  1114
  -------------------------  --------------------------------
  DS (det_simulation)                     1039         (93.27%)
  UU (unused)                               30         ( 2.69%)
  TI (tied)                                  4         ( 0.36%)
  RE (redundant)                             3         ( 0.27%)
  AU (atpg_untestable)                      38         ( 3.41%)
-------------------------------------------------------------
Coverage
  -------------------------
  test_coverage                                          96.47%
  fault_coverage                                         93.27%
  atpg_effectiveness                                    100.00%
-------------------------------------------------------------
#test_patterns                                             130
  #clock_sequential_patterns                               130
#simulated_patterns                                        256
CPU_time (secs)                                            0.5
-------------------------------------------------------------
Multiple Detection Statistics
-------------------------------------------------------------
  Detections                          DS Faults    Test Coverage
     (N)                        (Detection == N) (Detection >= N)
----------------------------  ---------------- ----------------
      1                              0 (  0.00%)   1039 ( 96.47%)
      2                              0 (  0.00%)   1039 ( 96.47%)
      3                             93 (  8.35%)   1039 ( 96.47%)
      4                             56 (  5.03%)    946 ( 87.84%)
      5                             54 (  4.85%)    890 ( 82.64%)
      6                             56 (  5.03%)    836 ( 77.62%)
      7                             50 (  4.49%)    780 ( 72.42%)
      8                             46 (  4.13%)    730 ( 67.78%)
      9                             54 (  4.85%)    684 ( 63.51%)
     10+                           630 ( 56.55%)    630 ( 58.50%)
-------------------------------------------------------------
bridge_coverage_estimate                                  94.71%
-------------------------------------------------------------
```

Example 4

The next example performs ATPG with the path delay fault model and only robust detection patterns. The example then repeats the pattern creation run after enabling the tool to use both robust and non-robust detection patterns. Be aware that when you change the type of test the tool can use to detect path delay faults, the tool automatically resets all AU faults identified in the preceding pattern creation run.

```
set_system_mode analysis
set_fault_type path_delay -robust_detection_only
read_faults my_path_delay_faultlist
create_patterns
...
set_fault_type path_delay -no_functional_detection
create_patterns
```

Example 5

The following example generates launch-off-shift patterns by holding all PIs:

```
set_pattern_type -sequential 0 -ram_sequential off
add_input_constraints -hold -all
set_fault_type transition -apply_hold_pi_attribute_to_shift_launch on
create_patterns
```

Example 6

The following example sets the UDFM as the current fault type:

```
set_fault_type udfm
```

Related Commands

add_faults
delete_faults
identify_redundant_faults
read_fault_sites
report_faults
set_fault_mode
set_fault_protection
set_multiple_detection
write_faults

set_file_compression

Context: unspecified, all contexts

Mode: all modes

Usage

set_file_compression [ON | OFf]

Description

Controls whether the tools read and write files with .Z or .gz extensions as compressed files (the default).

Files that contain large pattern sets consume a very large amount of disk space. Fault lists and the design data itself also take up a lot of disk space. To conserve this space, the tools normally store files in one of two compressed formats when you provide a filename with the appropriate extension, as follows:

- ".Z" specifies to compress the file using the Linux compress command.

Linux

Some versions of Linux do not support the compress command. In these cases, the tool will issue a warning, use the GNU gzip command to compress the file(s), and change the extension of the file being written to .gz. The tool can still read .Z files using zcat; however, Mentor Graphics recommends you use the .gz extension when writing compressed files.

- ".gz" specifies to compress the file using the GNU gzip command. You can control the type of GNU compression with the set_gzip_options command.

When compressed file handling is enabled and you provide a filename with either of the above extensions, the tool will automatically decompress (for reading) or compress (for writing) the specified file.

The set_file_compression command allows you to turn off the tool's normal compressed file handling functionality. This is useful in rare cases where files have either of the compressed file extensions, but should not be saved or read as compressed files.

Arguments

- ON

  An optional literal that enables compressed file handling. This is the default.

- OFf

  An optional literal that disables compressed file handling. When set to off, the tools process .Z or .gz files without using compression.

Example 1

Suppose the file *testpat.ascii.gz* is not a compressed file. The following example disables compressed file handling so the tool will read *testpat.ascii.gz* as a normal file rather than as a compressed file:

```
set_file_compression off
```

Example 2

The following example re-enables compressed file handling, then saves the file *fault.pat* in GNU format:

```
set_file_compression on
write_patterns fault.pat.gz
```

set_flattener_rule_handling

Context: unspecified, all contexts

Mode: setup

Usage

set_flattener_rule_handling ***rule_id*** [**Error** | **Warning** | NOTe | **Ignore**] [Verbose | Noverbose]

Description

Specifies how the tool globally handles flattening violations.

The set_flattener_rule_handling command specifies the handling of the messages for net checking, pin checking, and gate checking. You can specify that the violation messages for these checks be either error, warning, note, or ignore. If you do not specify error, warning, note, or ignore, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the initial invocation setting as specified in the following list of rules.

Each rules violation has an associated occurrence message and summary message. The tool displays the occurrence message only for either error conditions or if you specify the Verbose option for that rule. The tool displays the rule identification number in all rules violation messages.

Arguments

- ***rule_id***

  A required, non-repeatable literal that specifies the identification of the exact flattening rule violations whose message handling you want to change.

  The flattening rule violations and their identification literals are divided into the following three groups: net, pin, and gate rules violation IDs.

  - Net flattening violations are described in sections “FN1” through “FN9”.
  - Pin flattening violations are described in sections “FP1” through “FP13”.
  - Gate flattening violations are described in sections “FG1” through “FG8”.

- Error

  An optional literal that specifies for the tool to display both the error occurrence message and immediately terminate the rules checking.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Warning

  An optional literal that specifies for the tool to display the warning summary message indicating the number of times the rule was violated. If you also specify the Verbose option, the tool displays the occurrence message for each occurrence of the rules violation.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOTe

  An optional literal that specifies for the tool to display the summary message indicating the number of violations for that rule. If you also specify the Verbose option, the tool also displays the occurrence message for each occurrence of the rules violation.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Ignore

  An optional literal that specifies for the tool to not display any message for the rule's violations. The tool must still enforce some rules and they must pass to allow certain functions to be performed later.

  If you do not specify the Error, Warning, Note, or Ignore option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- NOVerbose

  An optional literal that specifies for the tool to display the occurrence message only once for the rules violation and give a summary of the number of violations.

  If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

- Verbose

  An optional literal that specifies for the tool to display the occurrence message for each occurrence of the rules violation.

  If you do not specify the Noverbose or Verbose option, then the tool uses either the handling from the last set_flattener_rule_handling command or, if you have not changed the handling, the tool uses the initial invocation setting.

Example

The following example changes the handling of the FG7 flattening rule to warning and specifies that each occurrence should be listed:

```
set_flattener_rule_handling fg7 warning -verbose
```

Related Commands

report_flattener_rules

set_drc_handling

set_gate_level

Context: unspecified, all contexts

Mode: all modes

Usage

set_gate_level **Primitive** | **Design**

Description

Specifies the hierarchical level of gate reporting and displaying.

The set_gate_level command specifies the hierarchical gate level at which the tool operates. This includes the reporting and schematic display of gate information. Once you set the gate level, the tool processes all subsequent commands using the new gate level.

Whenever you issue a command which invalidates the flattened model, the tool also invalidates the hierarchical gate display structure. This causes DFTVisualizer to clear the schematic view area. You can rebuild the hierarchical gate structure by creating a new flattened model. To do so, either enter and exit the Setup mode, or use the create_flat_model command.

Arguments

- **Primitive**

  A literal that specifies to display gate information at the built-in primitive gate level.

- **Design**

  A literal that specifies to display gate information at the design library hierarchical gate level. These are the top-level cells of the design library which are instantiated in your design. This is the default upon invocation of the tool.

Note

DFTVisualizer shows gate identification numbers (ID#s) with displayed instances only if the design level is set to Primitive.

Example

The following example sets the gate report level so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the Setup system mode):

```
set_system_mode analysis
set_gate_level primitive
set_gate_report error_pattern
report_gates i_1006/o
```

Related Commands

create_flat_model
set_gate_report
report_gates

set_gate_report

Context: all contexts

Mode: all modes

Usage

When EDT is Off

set_gate_report **Normal** | **SImulation_context** | **Trace** | **Error_pattern** |
{**PATtern_index** ***pattern_index*** [-Internal | -External] [-SCan_test | -CHain_test]} |
VCD ***time_value*** | {**Parallel_pattern** ***pattern_number***} |
Fault_status | **Test_data** | **TIe_value** | **COnstrain_value** | **SEq_depth_data** |
{**CLock_cone** ***pin_name***} |
{**CAPTURE_PROCedure** ***named_capture_procedure***} |
{**Drc_pattern** {{**Test_setup** | **TEST_End}** [{-Cycle | -Time} *n1* [*n2* | End]]} |
{**Load_unload** | **SHIft** | **SKew_load** | **SHADOW_Control** |
Master_observe | **SHADOW_Observe** | **STate_stability** | **STABle_after_setup** |
STABLE_Capture | **STABLE_Load_unload** | **STABLE_Shift**
[time | -All] } } |
{**DELay_data** {**LOcal_delay_data** |
STATIC_Arrival_time | **STATIC_PRopagation_time** | **STATIC_Slack** |
STATIC_PAth_delay |
ACTUAL_Arrival_time ***pattern_number*** [-Internal | -External] |
ACTUAL_PRopagation_time ***pattern_number*** [-Internal | -External] |
ACTUAL_Slack ***pattern_number*** [-Internal | -External] |
ACTUAL_PAth_delay ***pattern_number*** [-Internal | -External] } }
[-Timing_exceptions {ON | OFf}] [-MAX_Fanouts { *number* | UNlimited }]

When EDT is On

set_gate_report **Normal** | **SImulation_context** | **Trace** | **Error_pattern** |
{**PATtern_index** ***pattern_index*** [-Internal | -External] [-SCan_test | -CHain_test]} |
{**Parallel_pattern** ***pattern_number***} |
Fault_status |
Test_data |
TIe_value |
COnstrain_value |
SEq_depth_data |
{**CLock_cone** ***pin_name***} |
{**CAPTURE_PROCedure** ***named_capture_procedure***} |
{**Drc_pattern** {{**Test_setup** | **TEST_End**} [{-Cycle | -Time} *n1* [*n2* | End]]} |
{Load_unload | **SHIft** | **SKew_load** | **SHADOW_Control |**
Master_observe | **SHADOW_Observe** | **STate_stability** | **STABle_after_setup** |
STABLE_Capture | **STABLE_Load_unload** | **STABLE_Shift**
[time | -All]} |
{**K19** [-Initial | -Post_capture | -Low_power lp_sim#]
[-All | Load_unload | {Shift [-All | *cycle* [*cycle*]]}]} |

{**K22** [-Mask {No_mask | *mask#*}] [-All | Load_unload |
{Shift [-All | *cycle* [*cycle*]]}] |
F_rule_name}
{**DELay_data** {**LOcal_delay_data** |
STATIC_Arrival_time | **STATIC_PRopagation_time** | **STATIC_Slack** |
STATIC_PAth_delay |
ACTUAL_Arrival_time ***pattern_number*** [-Internal | -External] |
ACTUAL_PRopagation_time ***pattern_number*** [-Internal | -External] |
ACTUAL_Slack ***pattern_number*** [-Internal | -External] |
ACTUAL_PAth_delay ***pattern_number*** [-Internal | -External] } }
[-Timing_exceptions {ON | OFf}] [-MAX_Fanouts { *number* | UNlimited }]

For dft -scan

set_gate_report **Normal** |
Trace |
Error_pattern |
TIe_value |
Constrain_value |
{**Drc_pattern** {{**Test_setup** [{{-Cycle | -Time} *n1*} [*n2* | End]]} |
{**Load_unload** | **SHIft** | **SKew_load** | **SHADOW_Control** |
Master_observe | **SHADOW_Observe** | **STate_stability** |
STABle_after_setup | **STABLE_Capture** | **STABLE_Load_unload** |
STABLE_Shift} [-All | *time*]} [-MAX_Fanouts { *number* | UNlimited }]

Description

Specifies the information displayed by the report_gates command.

The set_gate_report command controls the type of additional information that the report_gates command displays. This command also controls the information displayed for each instance in the DFTVisualizer Debug window. When you exit the Setup system mode, the trace and any rules-checking error pattern results will not be available with this command. For information on the format output by the different options in this command, refer to the report_gates command description.

Note

When EDT is On, the tool does not explicitly simulate the EDT logic during scan loading and capture cycles. Therefore, report_gates cannot report simulation values for the EDT logic during these cycles regardless of the option set with the set_gate_report command.

Arguments

- **Normal**

  A literal that specifies for the report_gates command to display only its standard information. This is the default mode upon invocation of the tool.

- **simulation_context**

  A literal that specifies for the report_gates command to display the simulated values from the current simulation context. Refer to "Example 11" on page 1766.

- **Trace**

  A literal that specifies for the report_gates command to display the simulated values of the gates during the scan chain tracing. The format of these values depends on the contents of the shift procedure. If the shift procedure contains additional frames, the additional frames will also be displayed in the gate report data. The trace data relates to the simulation performed during the scan chain tracing. Use the Trace option to determine why a scan chain was not properly sensitized during the shift procedure.

  For an example of the information displayed, refer to "Example 2" on page 1755.

- **Error_pattern**

  A literal that specifies for the report_gates command to display the simulated value of the gates and its inputs for the pattern at which an audit error occurred.

- **PATtern_index** ***pattern_index*** [-Internal | -External] [-SCan_test | -CHain_test]

  A literal, integer, and optional switch triplet that specifies the pattern the report_gates command uses when displaying the value of a gate. The *pattern_index* must be a non-negative integer.

  When reporting a sequential element, the gate report also displays in a pair of brackets ([ ]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

  The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

  Glitch Reporting

  When a leading-edge DFF clock is gating a leading-edge DFF structure, the tool displays displays the glitch on the CLK line as 0(1)00 and lets the downstream DFF capture. For example:

```
//  /sff9_pulse_inter (366)  DFF
//      "S"    I  (000-000)  188-
//      "R"    I  (000-000)  32-
//      CLK    I  (010-0(1)00)  298-/AND02_pulse_inter/Y
//      "D0"   I  (000-111) 284-
//      "OUT"  O  (100-011-[1])  152-  153-
```

  The glitch 0(1)00 on the clock only shows up as the input pin values of the state element. The glitch does not show up as the output value of the clock driver.

Primitive or Design Level Gate Reporting

Depending on whether the tool is set up for primitive or design level gate reporting, you may see additional information as follows (use the report_environment command to check the current gate level):

- Primitive level (set_gate_level Primitive):

  If the sequential element you report on is part of a scan cell and the value captured by the element is not what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value and explanatory text is included indicating which element *is* observed for that scan cell and that pattern. “Unobs” and extra explanatory text is displayed, for example, if you report on the slave latch within an LSSD scan cell and the test procedure file includes a master_observe procedure (the slave is not observed when a master_observe procedure exists).

Note

When reporting gates at the primitive level, the tool displays captured/unload values only for the outputs of DFF and LA primitives, both scan and non-scan. Captured/unload values are not reported for transparent latches (TLAs).

- Design level (set_gate_level Design):

  If the sequential element you report on is a scan cell and an output value resulting from capture does not correspond to what is observed/unloaded from the scan cell for that particular pattern, the display shows “Unobs” next to the captured value. Explanatory text is not displayed for these “Unobs” values.

For examples of captured/unload displays for different scan cells and reporting levels (primitive or design), see the Examples section.

If, after setting the gate report with this option, you alter the tool environment, be sure to reissue the “set_gate_report” command. This will cause the tool to re-simulate the patterns, ensuring the values reported for a gate are up to date with the latest tool settings.

You can select from four pattern sources as specified here:

-Internal — An optional switch that specifies use of the tool’s internal pattern set. This is the default.

-External — An optional switch that specifies use of the tool’s external pattern set.

-Scan_test — An optional switch that specifies the report of the scan test pattern.

-Chain_test — An optional switch that specifies the report of the chain test pattern.

Together with the *pattern_index* argument, your option selection specifies the exact pattern to be reported as shown in Table 7-41.

Table 7-41. Pattern Sources

Display Pattern Type	scan_test	chain_test
internal	internal scan pattern (default)	internal chain pattern
external	external scan pattern	external chain pattern

- **VCD *time_value***

  A literal and numerical pair that sets the gate report source to the VCD file used with the analyze_simulation_mismatches command. The Verilog simulation values after the specified simulation *time_value* can be seen using the report_gates command.

  The possible reported values are:

  0 — Logic 0

  1 — Logic 1

  Z — Tristate

  X — Unknown

  U — Unused

  . — No VCD data

- **Parallel_pattern *pattern_number***

Note

Use this option only if another command directs you to use it; for example, the analyze_bus command might transcript a message that explicitly says to use "set_gate_report parallel_pattern 0". If you use this option, be sure report_gates is the first command you enter after simulation; bogus simulation values may be reported if you enter any other tool command after simulation, but before you report_gates. The Pattern_index argument is the recommended way to specify a pattern for the report_gates command to use when displaying the value of a gate.

A literal and integer pair that specifies the pattern number from the last simulation pass that you want the report_gates command to use when displaying the value of a gate. The *pattern_number* must be an integer between 0 and 63.

When reporting a sequential element, the gate report also displays in a pair of brackets ([ ]) at each output of the element. For non-scan elements, this is the value that resulted from capture and always matches the simulation value in the last frame. For scan elements, this value is the scan cell unload value, which may be different than the capture value (for example, for EDT masking values or cell constraints). For a multicycle pattern, the captured value is reported for the last cycle only, since for cycles that precede the last cycle, the captured value is already shown in the last frame.

The bracketed value is useful because it gives you a direct view of the results of good machine simulation, which sometimes changes the value captured in the last cycle such as the examples mentioned in the previous paragraph.

- **Fault_status**

  A literal that specifies for fault detection status of all gates to be preserved. Subsequent commands that cause the gate to be displayed will annotate the pins with fault status data. If a schematic is currently displayed in the DFTVisualizer Debug window and you change the gate report data (by issuing the set_gate_report command), all fault sites will be annotated with their fault detection status.

  Note that this switch is not valid for the bridge logic fault model.

  The format of the fault status data is as follows:

  <sa0-status:sa1-status> — for stuck-at faults
  <slow_to_fall-status:slow_to_rise-status> — for transition faults

  where sa0-status and sa1-status are one of the following:

  DS — Detected by simulation

  DI — Detected by implication

  PU — Possible detect untestable

  PT — Possible detect testable

  AU — Atpg untestable

  UC — Undetected uncontrolled

  UO — Undetected unobserved

  UU — Untestable unused

  BL — Untestable blocked

  TI — Untestable tied

  RE — Untestable redundant

  F — Recognized fault site, but no fault has been added yet

  N — Site is nofaulted; either because the fault site is inside a library model and internal faults are off, or because a user has nofaulted it with the add_nofaults command

  "- " — Nothing known about this pin; used for pins created by the flattening process or pins that are not fault sites (for example, the pins of an unnamed internal instance of a library cell)

  When you use the Fault_status option, subsequent report_gates commands display the fault sub-class for untestable faults (AU/UC/UO). For a complete list of these fault sub-classes, see Table 7-49.

  The DFTVisualizer command, report_display_instances, will report the fault detection status in the tool transcript.

- **Test_data**

  A literal that specifies for the report_gates command to display previously-calculated control and observe values. To set the Test_data option, you must do so prior to rules checking, and you must re-execute the design rules checking process; otherwise, no data (-) from this option will be available for gate reporting.

  This option is primarily for logic BIST purposes (when inserting control and observation points). The data for each pin of a reported gate consists of three integers indicating how many times the pin was controllable to 0, how many times it was controllable to 1, and how many times it was observable during the preceding analysis. The data is displayed in the following format:

  (# of times controlled to 0-# of times controlled to 1, # of times observed)

Note

When the Test_data option is in effect, the report_environment command will show "BIST data" as the current gate report setting.

- **TIe_value**

  A literal that specifies for the report_gates command to display the simulated values that result from all natural, tied gates and learned constant value non-scan cells.

  Tie_value is available at any time the flattened model exists. The value displayed on the TIE node is set by simulation. All internal nodes are initially set to X for simulation. Subsequently, ATPG or various commands may put values on the nodes. Occasionally, a set of circumstances may occur where an X shows up on a TIE node.

  During operation, ATPG typically marks for simulation only those gates which are of interest for sensitization and propagation of a particular fault. If a gate is not marked, it will not be simulated, and any nodes associated with it may remain at an "X" value, even though they are connected to a TIE gate.

  While tie_value reports the status of node values once it comes out of the DRC checks, the constrain_value tells you what the gate is suppose to act like for future consideration.

- **Constrain_value**

  A literal that specifies for the report_gates command to display the simulated values that result from all natural tied gates, learned constant value non-scan cells, constrained pins, and constrained cells.

  The report_gates command displays three values that are separated by a slash (/). These values are the gate constrained value (0, 1, X, or Z), the gate forbidden values (-, 0, 1, Z, or any combination of 01Z), and the fault blockage status (- or B, where B indicates all fault effects of this gate are blocked).

  The displayed values are only available at the end of DRC once the tool successfully leaves Setup Mode. While tie_value reports the status of node values once it comes out of the DRC checks, the constrain_value tells you what the gate is supposed to act like for future consideration.

- **SEq_depth_data**

  A literal that specifies for the report_gates command to display the calculated gate sequential depths.

  When you select a non-zero sequential depth, the tool performs a learning process to identify the minimum depths necessary to satisfy controllability and observability requirements for all gates using the current clock_sequential cells. The tool calculates both the 0-state and 1-state controllability depths. At the end of the analysis, the tool displays a summary message that indicates the largest sequential test depth, along with the largest control and observe depth. The test generator uses the sequential depth information in making decisions and avoiding paths whose sequential depth exceeds the maximum allowed sequential depth.

  The report_gates command includes three values separated by a comma and a dash. The first value is the 0-state controllability depth, the second value is the 1-state controllability depth, and the third value is the observability depth. The maximum reported depth is 255. If controllability or observability is logically impossible or exceeds 255, the report displays an asterisk (*) in the corresponding fields.

- **Clock_cone *pin_name***

  A literal and string pair that specifies the clock pin for which the report_gates command displays the clock cone data.

  The clock cone data from the report_gates command is the same data that is available as error data for clock rules violations. You can only use this option after flattening the simulation model using the create_flat_model command.

  The *pin_name* must be a valid clock pin or an error condition occurs. The tool considers the pin equivalents when calculating the clock cones. State elements that the tool identifies as capturing on the clock's trailing edge will not propagate the clock effect cone. During the Setup system mode, this information is not available and the tool assumes all state elements capture with the leading edge of the selected clock.

- **CAPTURE_PROCedure *named_capture_procedure***

  A literal that reports the value at each gate implied by forced and conditional assignments defined in the named capture procedure (NCP). This is useful for debugging NCP-specific issues like the inability to detect a fault or prevent bus contention.

- **Drc_pattern *procedure_name*** [-<u>All</u> | *time*]

  Two literals and an optional time triplet that specifies the name of the procedure and the time in the test procedure file that the report_gates command uses to display a gate's simulated value.

  The valid literal choices for the *procedure_name* option (for use with Drc_pattern) are as follows:

  Test_setup — A literal that specifies the use of the test_setup procedure. In the test procedure file, this procedure sets non-scan elements to the state you desire for the **load_unload** procedure. The tool uses the entire test_setup procedure unless you

restrict the report to a certain portion using the -Cycle or -Time switch. In order to conserve screen width, time values are listed vertically in Test_setup gate reports.

TEST_End — A literal that specifies the use of the test_end procedure. In the test procedure file, test_end is used to add a sequence of events to the end of a test pattern set. The tool uses the entire test_end procedure. In order to conserve screen width, time values are listed vertically in test_end gate reports.

-Cycle — A switch that indicates *n1* is a cycle number and specifies to start the report at cycle *n1*.

-Time — A switch that indicates *n1* is a time and specifies to start the report at time *n1*. The time units are based on the timescale defined in the test procedure file, which by default is 1 nanosecond.

n1 — An integer that specifies a cycle or time at which to start reporting. When used with the -Time switch, *n1* specifies a time.When used with the -Cycle switch, *n1* specifies a cycle. The tool numbers cycles beginning with 0; so, for example, to specify the second cycle, you would use “-cycle 1”.

n2 — An optional integer that specifies to report from cycle (or time) *n1* to cycle (or time) *n2* and stop reporting.

End — An optional literal that specifies to report from cycle (or time) *n1* to the end of the test_setup procedure.

Note

When using the -Cycle or -Time switch, if you do not include either the *n2* or End argument, gate reports will show data for only the *n1* cycle (or time).

Load_unload — A literal that specifies the use of the load_unload procedure. The test procedure file must contain this procedure, which describes how to load and unload data in the scan chains.

SHIft — A literal that specifies the use of the shift procedure. The test procedure file must contain this procedure, which describes how to shift data one position down the scan chain.

SKew_load — A literal that specifies the use of the skew_load procedure. In the test procedure file, this procedure describes how to propagate the output value of the preceding scan cell into the master memory element of the current cell (without changing the slave), for all scan cells.

SHADOW_Control — A literal that specifies the use of the shadow_control procedure. In the test procedure file, this procedure describes how to load the contents of a scan cell into the associated shadow.

Master_observe — A literal that specifies the use of the master_observe procedure. In the test procedure file, this procedure describes how to place the contents of a master into the output of its scan cell.

SHADOW_Observe — A literal that specifies the use of the shadow_observe procedure. In the test procedure file, this procedure describes how to place the contents of a shadow into the output of its scan cell.

STate_stability — A literal that specifies the display of the simulation values for the **load_unload** procedure and the capture clock cycle that the tool used to determine the constant value state elements at the initial load time. The report separates the **shift** procedure values, **load_unload** procedure values, and the capture clock cycle with parentheses. This format provides information that is helpful when you are trying to debug boundary scan.

Note

Only the simulation values for the first **load_unload** procedure that follows the **test_setup** procedure are displayed. Use the load_unload literal to display the final stable values in the **load_unload** procedure.

The state_stability option is not a procedure.

For detailed information about state stability reporting, refer to the "State Stability Reporting" appendix in the *Tessent Scan and ATPG User's Manual*.

STABLE_Capture — A literal that specifies the display of the values that are constant throughout the capture procedures.

0, 1, or Z — Indicates a constant value exists throughout the capture cycles.

X — Indicates a constant value does not exist throughout the capture cycles.

STABLE_Load_unload — A literal that specifies the display of the values that are constant throughout the load_unload cycles.

0, 1, or Z — Indicates a constant value exists throughout the load_unload cycles.

X — Indicates a constant value does not exist throughout the load_unload cycles.

STABLE_Shift — A literal that specifies the display of values that are constant throughout the shift cycles.

0, 1, or Z — Indicates a constant value exists throughout the shift cycles.

X — Indicates a constant value does not exist throughout the shift cycles.

STABle_after_setup — A literal that specifies the display of the values that are constant in the period following the end of the test_setup procedure and the beginning of the test_end procedure. For scan patterns, the value is constant throughout scan and capture cycles. For ijtag patterns, the value is constant throughout any ijtag operations.

If a load_unload procedure exists, the values established at the end of test_setup are verified for stability using the add_input_constraints -c1 and c0 values which are also forced to a constant value during the load_unload procedure.

If a load_unload procedure does not exist, as in the ijtag context, stability analysis is only performed with the add_input_constraints -c1 and -c0 values.

0, 1, or Z — Indicates a constant value exists at the end of test_setup and is maintained for all subsequent events up to the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

X — Indicates a constant value does not exist in the period following the end of test_setup and the beginning of test_end, if one exists, or through the end of the test if test_end does not exist.

-All — An optional switch that specifies use of all times in the test procedure file. This is the default.

time — An optional positive integer, greater than 0, that specifies a time in the test procedure file.

This subsection describes Tessent FastScan and Tessent TestKompress gate reporting when Drc_pattern specifies a procedure.

When you set the gate report to a DRC pattern and procedure, most of the data reported by the report_gates command is simulation data regarding the specified procedure. For example, if you specify the load_unload procedure, you will get simulation data regarding the load_unload procedure immediately following the test_setup procedure. Statements like **apply shift** are broken out by surrounding ()s.

The last group of data is more specialized. Its contents depend on the capture clock being set with the -atpg switch. The starting state for this simulation results from simulating the events of the test setup procedure, followed by the load_unload procedure and its apply procedures (shift and shadow_control).

- Case 1: No Capture Clock

  There will be 1 or 2 values in the last pair of ()s. The first value is the simulation state that results from holding all PIs at their pin constrained value and setting all clocks to X at the end of load/unload.

  If any state element has a different binary value than the one it had at the end of simulating test setup, its value will be changed to X, the affect propagated, and the final values saved in the second value between the ()s. See the following examples:

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 100;
   pulse myclk 200 100;
   period 400;
end;

procedure shift =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force_sci;
      measure_sco;
      pulse myclk;
   end;
end;
```

```
procedure load_unload =
   scan_group grp1;
   timeplate gen_tp1;
   cycle =
      force scan_en 1;
      force myclk 0;
   end;
   apply shift 2;
end;

report_gates clk
//  /CLK  primary_input
//      CLK    O  (0)(0X0)(0)(X)
```

The first (0) is the simulation of the events in the load_unload procedure prior to the apply shift, (0X0) is the simulation of the shift procedure, (0) is the simulation of the events in the load_unload after the shift, and finally (X) is the simulation of the clocks at X.

- Case 2: Capture Clock -Atpg

  There will be 3 or 4 values in the last pair of ()s. The first three values result from simulating a pulse of the capture clock with all other clocks set to the off value.

  If any state element has a different binary value than the one it had at the end of simulation test setup, its value is changed to X, the effect is propagated, and the final values are saved in the fourth value between the ()s.

K19 [-Initial | -Post_capture | -Low_power lp_sim#]
[-All | Load_unload | {Shift [-All | cycle [cycle]]}]
(Only available when EDT is On)

Note

Use this argument only when necessary. For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, the Drc_pattern argument will be sufficient and less time-consuming than this one.

A literal and an optional, procedure pair that specifies to include a gate's simulation data resulting from the K19 design rules check (DRC) in the output displayed by the report_gates command. If you specify "K19" without any options, *all* K19 simulation values for every event specified in the test procedure file are displayed except test_setup events.

Note

The test_setup simulation data is available after you use the "set_gate_report drc_pattern test_setup" command.

The options enable you to limit the content of the display to the data for a certain procedure and/or for a certain cycle or range of cycles. For a detailed explanation of

the K19 DRC, refer to "K19" and to "Understanding K19 Rule Violations" in the *Tessent TestKompress User's Manual*.

K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K19 reporting when you need to see the specific simulation values applied in each shift cycle. This is not commonly required as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

You must set this option in Setup system mode. Also, DRC handling for the K19 rule must be set to either error (the default) or warning, but not to ignore.

The following three switches specify to report simulated and emulated (where applicable) values for selected gates as described here:

-Initial — An optional switch that specifies to report the very first K19 simulation result.

-Post_capture — An optional switch that specifies to report the second K19 simulation result (after capture); if low-power hardware exists and is enabled, its first simulation result is reported.

-Low_power lp_sim# — An optional switch that specifies to report the first or second simulation result for the low-power hardware, if one exists and is enabled. The lp_sim# argument can have a value of either 1 or 2.

The following three switches specify the use of the events in the test procedure file.

-All — An optional switch that specifies use of all events in the test procedure file except test_setup events. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle — A positive integer corresponding to a particular shift cycle. If you specify a single *cycle* number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all cycles in the range. Cycles are counted from 1 and include independent shifts.

K22 [-Mask {No_mask | mask#}] [-All | Load_unload | {Shift [-All | cycle [cycle]]}]
(Only available when EDT is On)

Note

Use this argument only when necessary. For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, the Drc_pattern argument will be sufficient and less time-consuming than this one.

An optional literal and procedure pair that include the gate simulation data resulting from the K22 design rules check (DRC) in the output displayed by the report_gates

command. If “K22” is specified without any options, the K22 simulation values for every event specified in the test procedure file display. The options enable you to limit the content of the display to just the data associated with a certain procedure and/or to a certain cycle or range of cycles. For a detailed explanation of the K22 DRC, refer to “K22” and to “Understanding K22 Rule Violations” in the *Tessent TestKompress User’s Manual*.

K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting, as the tool has to log simulation data for all simulated setup and shift cycles. Use K22 reporting when debugging incorrect compactor mask settings for specific patterns simulated by the design rule check. This is not commonly required, as most problems are usually due to setup problems, such as path sensitization and setting of control signals, that are usually not pattern-specific or cycle-specific.

Note

The K22 evaluations of compactor gates are synchronized with measure_sco statements in the test procedure file.

You must set this option in Setup system mode. Also, DRC handling for the K22 rule must be set to either error (the default) or warning.

-All — An optional switch that specifies use of all events in the test procedure file. This is the default.

Load_unload — An optional literal that specifies use of the load_unload procedure.

Shift — An optional literal that specifies use of the shift procedure.

-All — An optional switch that specifies use of all shift cycles. This is the default.

cycle — A positive integer corresponding to a particular shift cycle. If you specify a single *cycle* number, data for only that cycle is displayed. If you provide a second integer, separated from the first by a space, the tool interprets the two integers as the endpoints of a range of cycles and displays the data for all the cycles in the range. Cycles are counted from 1 and include independent shifts.

-Mask — An optional switch that sets the masking mode for the simulation data.

No_mask — A literal that together with -Mask specifies to report only non-masking mode simulation data.

mask# — A positive integer that, together with -Mask, specifies the identification number of a mask. The tool assigns a number to a mask according to the core scan chain the mask enables the tool to observe. Mask 3, for example, enables observation of the third core scan chain. This option thus limits reporting to the simulation data for the scan chain made observable by the specified mask.

F_rule_name — A literal that specifies an F rule between F1-F21.

Note, you must specify a specific occurrence of a F20 violation using the string “F20-occurence#” where occurence# is the violation occurrence . For more information, see the report_drc_rules command.

- **DELay_data** [delay_data_option]

  A required literal pair that specifies to report delay data when using the report_gates command. See "Using Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

  Reported delay data is one of the following data types:

  Number — Indicates delay data values.

  STAT (Static) — Indicates that no transition is possible on the pin so a delay value is not calculated or reported.

  NA (Not Applicable) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Static timing information)

  - (dash) — Indicates the pin is not in the detection path so a delay value is not calculated or reported. (Actual timing information) For the *local_delay_data* literal, a dash (-) is reported in the case of a conditional IO path delay; in this case, the timing information is included in the report as shown in Example 9.

  The following *delay_data_option* literals are available:

  LOCAL_delay_data

  Reports the local delay data, including the wire delay (INTERCONNECT) and gate delay (IOPATH) associated with the flattened gate pin.

  STATIC_Arrival_time

  Reports the longest static rising and falling arrival time of the gate and direct fan-ins. Both minimal and maximum arrival time will be reported as the following format:

  (max_rising_arrival_delay, max_falling_arrival_delay) /

  (min_rising_arrival_delay, min_falling_arrival_delay)

  STATIC_PRopagation_time

  Reports the longest static rising and falling propagation time of the gate and direct fan-ins.

  STATIC_Slack

  Reports the static test slack of the gate and direct fan-ins. The test slack is computed as the difference between the test clock period and the summation of longest arrival time and longest propagation time. The format of the reported data is:

  (test_slack_of_rising, test_slack_of_falling)

  STATIC_PAth_delay

  Reports the longest static rising and falling path delay time through the gates.

  ACTUAL_Arrival_time [-Internal | -External]

  Reports the actual arrival time of the gate and direct fan-ins. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum arrival time of given cycle will be reported. In addition to the actual arrival time, the simulation value of the specified pattern is also reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set. This is the default for Tessent TestKompress and Tessent FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_PRopagation_time [-Internal | -External]

Reports both the actual propagation time and simulation propagation time. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum propagation time of given cycle will be reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set. This is the default for Tessent TestKompress and Tessent FastScan.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Slack [-Internal | -External]

Reports both the actual and simulation slack values. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum slack of given cycle will be reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

-External — An optional switch that specifies use of the tool's external pattern set.

ACTUAL_Path_delay [-Internal | -External]

Reports both the actual and simulation path delay through the gates. This will first simulate the pattern specified by pattern_index. When there are more than two at-speed cycles, the maximum path delay among all cycles will be reported.

-Internal — An optional switch that specifies use of the tool's internal pattern set.

-External — An optional switch that specifies use of the tool's external pattern set.

Note

The calculations of arrival time and propagation time assume that all PIs change value at time 0. The delays at the clock port and the setup time of the sequential elements are ignored.

- -Timing_exceptions ON | OFf

  An optional switch and literal that turns on or off the timing exceptions (false paths or multicycle paths) information reporting for the report_gates command and the DFTVisualizer debug window. This option is only effective during non-Setup mode. The default is off.

  Figure 7-35 illustrates the false path definition point and the false path intersection cone and an effect cone.

Figure 7-35. False Path Intersection and Effect Cones

SDC File contains:

```
set_false_path -from [get_clocks {Clk1}] -to [get_pin {G3/A}]
```

In this figure, the false path begins from flops in clock domain Clk1 to internal pin G3/A. The false path's intersection cone (G_1) is the logic between all **-from** points and the **-to** point. The false path's effect cone (G_4 and U_3) is the combinational fanout cone of the **-to** point.

When you set the -Timing_exceptions switch to ON, any subsequent report_gates command can include one or more of the following false or multicycle path-related keywords in the report:

- **Fr** — Indicates the gate is a false or multicycle path from point.
- **To** — Indicates the gate is a false or multicycle path to point.
- **Th** — Indicates the gate is a false or multicycle path thru point.
- **In** — Indicates the gate is inside a false or multicycle path combinational intersection cone.
- **Ef** — Indicates the gate is in a false or multicycle path combinational effect cone.
- **--** — Indicates the gate is outside the false or multicycle path region.

The report can also include numbers, which indicate that a gate is part of at least one multicycle path. If there are two numbers, they indicate the minimum and maximum number of cycles. For example, "1,5" means the gate is on a multicycle path with minimum of 1 cycle and maximum of 5 cycles. If there is only one number, it means that the minimum and maximum number of cycles are the same. For example, "2" means the gate is on a 2-cycle

path. If there is no number, then it is either on a 1-cycle path (when "--" is shown), or on a false path (when at least one of Fr, To, Th, Ef, In is shown). The following shows how the numbers display in set_gate_report command output:

```
//  /I3/reg1   sff
//      D      I  (--)  /cnt[2]
//      SI     I  (--)  /I0/i0/Q
//      SE     I  (--)  /scan_en
//      CLK    I  (--)  /clock
//      Q      O  (In/1,5)  /I3/ix28/S0  /I3/ix30/S0  /I3/ix32/S0
//      QB     O  (--)  /I3/reg3/SI
```

The -Timing_exceptions argument does not reset the previous specified gate report option; consequently, you can use this argument in conjunction with other set_gate_report settings.Refer to "Example 8" on page 1763.

- -MAX_Fanouts { *number* | UNlimited }

  An optional switch paired with either a positive integer or keyword that specifies the maximum of fanout gates to be reported by the report_gates command. When you specify the Unlimited keyword, report_gates reports all fanout gates without truncation. The default is limited to 99 fanout gates. The tool issues a message if a reported gate drives more than the limit you specified with this command.

Example 1

The following example sets the gate report so that reporting and display show the simulated values of the gate and its inputs (assuming a rules checking error occurred when exiting the setup system mode):

set_system_mode analysis
set_gate_report error_pattern
report_gates i_1006/o

Example 2

The following example illustrates a shift procedure containing two clocks that are pulsed in sequence and the corresponding gate report display when the gate report is set to trace. The data displayed is with respect to the shift procedure.

```
procedure shift =
   force_sci     0;
   measure_sco   0;
   force clk   1 1;
   force clk   0 2;
   force tclk  1 3;
   force tclk  0 4;
   period        6;
end;
```

report_gates 32

```
//  /I_15 (32)  NOR
//      i0     I  (XXXXX)  16-/I_16/out
//      i1     I  (XXXXX)  17-/I_6/out
//      out    O  (XXXXX)  43-/S2
```

report_gates tclk

```
//  /TCLK (9)  PI
//      TCLK   O  (00010)  40-/I_13/clk
```

report_gates clk

```
//  /CLK (4)  PI
//      CLK    O  (01000)  20-/I_20/I_225/i0
21-/I_23/I_225/i0
```

Example 3

The following is sample output of the report_gates command for reporting path delay patterns.

set_system_mode analysis
create_patterns
set_gate_report pattern_index 0
report_gates /FF1

```
//  /FF1  SLSDFFQ0FF
//      CLK    I  (L0-L1-L0)  /P_CLK1
//      M2SO   I  (L0-L0-L0)  /M2SOi0/Q
//      D      I  (H1-HX-HX)  /I2
//      SI2M   I  (L0-L0-L0)  /SI2Mi0/Q
//      SI     I  (0-0-0)  /FF2/SO
//      MHN    I  (L1-L0-L1)  /MHNi0/Q
//      Q      O  (0-0-1)  /IN2/A
//      SO     O  (X-X-X)  /P_SO1
```

The Ls and Hs in the gate report refer to clock-off values. These are only displayed for path-delay faults. They represent the clock off value and can be either X, L, H, or Z. The tool reports the clock off state prior to the actual pin value only for pins that are on a clock path.

The clock path value is a value that appears on a gate when pin constraints are considered and when all clocks are at their defined off states. For example, assume your design has an OR gate where one input is connected to primary input pin A. A is constrained to value 1, and the other pin is connected to primary input pin CLK, which is defined as a clock with off-state 0. The clock path values, where Y is the output of the OR gate, would be:

```
A     H
CLK   L
Y     H
```

Now, assume A is a regular input pin; that is, it has no pin constraint and is not a clock. The clock path values now are:

```
A     X
CLK   L
Y     X
```

Y is in the clock path and if you use “set_gate_report clock_cone CLK”, you will see that Y is in the clock cone. Finally, assume that this gate is now an AND gate and A is still a regular input pin. The clock path values are:

```
A      X
CLK    L
Y      L
```

Since the clock path depends on both clocks and pin constraints, a pin that is in the clock path does not necessarily have to be in the clock cone.

An S followed by 0, 1, or X appears in patterns generated when you issue the command set_pattern_type -Multiple_load On. The S refers to these patterns being scan load operations.

Example 4

The following example shows sample output of the report_gates command when the report type is changed to report simulation data using the following test_setup procedure.

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 10;
   pulse clk1 20 10;
   pulse clk2 20 10;
   period 40;
end;

timeplate gen_tp2 =
   force_pi 10;
   measure_po 30;
   pulse clk1 60 10;
   period 100;
end;

procedure test_setup =
   scan_group grp1;
   timeplate gen_tp1;
   // First cycle (cycle 0), one event (force)
   cycle =
      force clk1 0;
      force clk2 0;
      force reset 0;
      force A 0;
      ...
      force E 0;
   end;
   // Second cycle, three events (force, pulse on, pulse off)
   cycle =
      force B 1;
      pulse reset;
      pulse clk2;
   end;
   // Third cycle, three events (force, pulse on, pulse off)
   cycle =
      force A 1;
      pulse clk2;
   end;
   // Fourth cycle, three events (force, pulse on, pulse off)
   cycle =
      timeplate gen_tp2;
      force A 0;
```

```
      pulse clk1;
   end;
end;
```

set_gate_level primitive
set_gate_report drc_pattern test_setup
set_system_mode analysis
report_gates /ff20

```
// /ff20 (54)  DFF
//    "IO"  I   15-
//    "I1"  I   14-
//    CLK   I   6-/clk2
//    D     I   7-/A
//    "OUT" O   24-

//    Cycle: 0   1   2   3
//    -----  -   --- --- ---
//                  11 111
//               467 801 389
//    Time:  0   000 000 000
//    -----  -   --- --- ---
//    "IO"   0   000 000 000
//    "I1"   0   000 000 000
//    CLK    0   010 010 000
//    D      0   000 111 000
//    "OUT"  X   X00 011 111
```

Cycle count starts at 0.

Force at cumulative time 40 (1st event of 2nd cycle).
Pulse off clk2 at cumulative time 70 (3rd event of 2nd cycle).
(Time is read vertically; time unit is based on timescale defined in test procedure file - 1 ns by default.)

Clock pulse on & output update are shown simultaneous.
(with zero delay)

set_gate_report drc_pattern test_setup -cycle 2 end
report_gates /ff20

```
//  /ff20 (54)  DFF
//    "IO"   I  15-
//    "I1"   I  14-
//    CLK    I  6-/clk2
//    D      I  7-/A
//    "OUT"  O  24-
//
// Cycle: 2   3
// ----- --- ---
//        11 111
//       801 389
// Time: 000 000
// ----- --- ---
// "IO"  000 000
// "I1"  000 000
// CLK   000 000
// D     111 000
// "OUT" 000 000
```

Example 5

The following examples show sample output of the report_gates command for sequential elements when the report type is set to report simulation data for a particular pattern. For more

information about what the tool displays for each cycle, refer to "Understanding Event Simulation for DFFs and Latches" in the *Tessent Scan and ATPG User's Manual*.

- Mux-DFF—primitive level

  The DFF captured 1 (shown in "[]") at the end of the cycles. This DFF is also the observe element for this scan cell. Therefore, the captured value is also the unload value.

set_gate_report pattern 6
set_gate_level primitive
report_gates /re1

```
//  /re1 (26)  MUX
//      TE     I  (000-000)  19-/scan_en
//      D      I  (000-000)  2-/d[1]
//      TI     I  (000-000)  80-/re0/Q
//      "OUT"  O  (000-000)  81-
//  /re1 (81)  DFF
//      "S"    I  (000-000)  20-
//      "R"    I  (000-000)  21-
//      CP     I  (000-010)  62-/rh1/CPB
//      "D0"   I  (000-111)  26-
//      Q      O  (000-011-[1]) 103-/sout[0]
//      MASTER  cell_id=0  chain=chain0  group=group0  invert_data=FFFF
//  Reported all gates in instance re1
```

- Mux-DFF—design level

set_gate_report pattern 6
set_gate_level design
report_gates /re1

```
//  /re1  FD1SQM1D
//      D      I  (000-000)  /d[1]
//      CP     I  (000-010)  /rh1/CPB
//      TI     I  (000-000)  /re0/Q
//      TE     I  (000-000)  /scan_en
//      Q      O  (000-011-[1]) /sout[0]
```

- Non-scan DFF—primitive level

set_gate_level primitive
report_gates /rh1/UDP1

```
//  /rh1/UDP1 (25)  INV
//      "I0"   I  (011-111)  79-
//      "OUT"  O  (100-000)  62-
//  /rh1/UDP1 (79)  TLA
//      "S"    I  (000-000)  20-
//      "R"    I  (000-000)  21-
//      "C0"   I  (010-000)  61-
//      "D0"   I  (111-111)  38-
//      "OUT"  O  (011-111)  25-
//  Reported all gates in instance rh1/UDP1
```

- Non-scan DFF—design level

set_gate_level design
report_gates /rh1/UDP1

```
//  /rh1  LCLKRHDELLD350D
//      CP     I  (101-111)  /inv0/Z
//      LD     I  (000-000)  /ld[0]
//      TE     I  (000-000)  /scan_en
//      CPB    O  (100-000)  /fout1  /inv1/A  /rh11/CP
//                           /nonscan_d_b/D  /re0/CP  /re1/CP
//                           /nonscan_d_b/TI
```

- LSSD scan cell (master latch and slave latch with master_observe)—primitive level

  If reporting the master, which is the observed element, the captured value is also the unload value. If reporting the slave, an “Unobs” appears beside the captured value to remind you that this captured value is not the unload value. Notice also the additional message at the end of the report for the slave, telling you that the master (217) is the observation point.

set_gate_level primitive
report_gates 217 218

```
//  /I26/L1 (217)  LA
//      "S"    I  (000-000)  26-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (010-000)  164-/clk_regen/I_crinv_c1/I2/OUT
//      "D0"   I  (111-111)  154-
//      SCANCLOCK  I  (000-000)  193-/clk_regen/I_inv_aclk3/I2/OUT
//      SCANIN  I  (111-111)  104-/testblock/msff_output/I12/OUT
//      Q      O  (111-111-[1])  33-/I26/I130/IN
//      MASTER  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//  /I26/L2 (218)  LA-IL
//      "S"    I  (000-000)  130-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (000-010)  203-/clk_regen/I_crinv_mc2/I2/OUT
//      DATA   I  (111-111)  33-/I26/I130/OUT
//      Q      O  (111-111-[1 Unobs])  34-/I26/I12/IN
//      SLAVE  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//      Latch is not the observation point for reported pattern.
//      Observation point is at MASTER gate 217.
//      Learned latch port types: IL
```

- Masked LSSD scan cell—primitive level

  If the cell is masked (unload value = X due to an XX cell constraint added with the add_cell_constraints command, for example), the tool informs you that the cell is not observed for this pattern.

set_gate_level primitive
report_gates 311

```
// /ix301/slave (311) LA
// "S"   I (000-000) 43-
// "R"   I (000-000) 206-
// "C0"  I (000-010) 116-
// "D0"  I (000-000) 59-
// "OUT" O (111-100 [X Unobs]) 58-
```

```
// SLAVE cell_id=5 chain=chain2 group=grp_lssd invert_data=TTFT
//    Scan cell not observed for this pattern.
```

- LSSD scan cell, master and slave with master_observe—design level

  Notice that the tool does not show explanatory text at the design level for “Unobs” cells.

set_gate_level design
report_gates 217 218

```
//  /I26/L1 (217)  LA
//      "S"    I  (000-000)  26-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (010-000)  164-/clk_regen/I_crinv_c1/I2/OUT
//      "D0"   I  (111-111)  154-
//      SCANCLOCK  I  (000-000)  193-/clk_regen/I_inv_aclk3/I2/OUT
//      SCANIN  I  (111-111)  104-/testblock/msff_output/I12/OUT
//      Q      O  (111-111-[1])  33-/I26/I130/IN
//      MASTER  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//  /I26/L2 (218)  LA-IL
//      "S"    I  (000-000)  130-
//      "R"    I  (000-000)  20-
//      CLOCK  I  (000-010)  203-/clk_regen/I_crinv_mc2/I2/OUT
//      DATA   I  (111-111)  33-/I26/I130/OUT
//      Q      O  (111-111-[1 Unobs])  34-/I26/I12/IN
//      SLAVE  cell_id=1  chain=sc0  group=g1  invert_data=FFFF
//      Latch is not the observation point for reported pattern.
//      Observation point is at MASTER gate 217.
//      Learned latch port types: IL
```

Example 6

The following example sets the gate report so that reporting and display show all the K19-simulated values of each reported gate:

```
set_gate_report drc_pattern k19
// Warning: Data will be accessible after running DRC.
```

The report_environment command would show the following for the gate report:

```
gate report = display K19 sim data, procedure(s): test_setup,
   load_unload, shift - all cycles
```

The following is sample output of the report_gates command when the report type is changed prior to DRC to report K19 simulation data for the following load_unload procedure and shift cycles 8 through 10. The example begins in Setup system mode.

```
timeplate gen_tp1 =
   force_pi 0;
   measure_po 1;
   pulse clk 2 1;
   pulse edt_clock 2 1;
   period 4;
end;

procedure load_unload =
   scan_group grp1;
   timeplate gen_tp1;
```

```
      cycle =
         force clk 0;
         force edt_bypass 0;
         force edt_clock 0;
         force edt_update 1;
         force scan_en 1;
         pulse edt_clock;
      end;
      apply shift 11;
   end;
   procedure shift =
      scan_group grp1;
      timeplate gen_tp1;
      cycle =
         force_sci;
         force edt_update 0;
         measure_sco;
         pulse clk;
         pulse edt_clock;
      end;
   end;

   set_gate_report drc_pattern k19 load_unload 8 10
   // Warning: Data will be accessible after running DRC.
   set_system_mode analysis
   report_gates /edt_i/edt_decompressor_i/ix159/Y

   // /edt_i/edt_decompressor_i/ix159 (142) NXOR
   //       A0 I 130-/edt_i/edt_decompressor_i/ix241/Y
   //       A1 I 84-/edt_i/edt_decompressor_i/lfsm_lockup_4/Q
   //       Y  O 154-/edt_i/ix90/A
   // Proc: ld_un ~ shi 8 sh 9 sh10
   // ----- ----- ~ ----- ---- ----
   // Time: i0234 ~ i0123 0123 0123
   // ----- ----- ~ ----- ---- ----
   // Sim:  XXX00 ~ 00000 0001 1110
   // Emu:  ----- ~ 0---0 ---1 ---0
   // No mismatch.
   // Monitor: EDT decompressor chain 1 output.
```

In the preceding report, consecutive cycles are listed from left to right, with a “~” indicating a break in the data. A blank space separates each cycle’s data from that of neighboring cycles. The numbers in the row labeled “Time” are the relative times of each event in a cycle as specified in the procedure file. An “i” indicates the start of a new sequence of events. Because the gate is one the tool monitors for the purpose of pinpointing where K19 errors first occur, the report shows the values the tool emulated for the gate and whether there were any mismatches between the emulated and simulated values.

Example 7

The following example sets the gate report so reporting and display use the K22 simulated values for the load_unload procedure of the preceding example and shift cycles 4-8 for the fourth core scan chain.

```
   set_gate_report drc_pattern k22 -mask 4 4 8
   // Warning: Data will be accessible after running DRC.
```

The report_environment command would show the following for the gate report:

```
gate report = display K22 sim data mask 4, procedure(s):
   load_unload, shift - range 4 - 8
```

The following is sample output of the report_gates command when the report type is changed prior to DRC. The example begins in Setup system mode.

```
set_gate_report drc_pattern k22 -mask 4 4 8
// Warning: Data will be accessible after running DRC.
set_system_mode analysis
report_gates /edt_i/edt_bypass_logic_i/ix7/Y

// /edt_i/edt_bypass_logic_i/ix7 (216) OR
//       "I0"  I 215-
//       B0    I 76-/edt_i/edt_bypass_logic_i/ix5/Y
//       Y     O 256-/edt_channels_out1
//
// Proc: ld_un ~ shi 4 sh 5 sh 6 sh 7 sh 8
// ----- ----- ~ ----- ---- ---- ---- ----
// Time: i 234 ~ i 123 123  123  123 123
// ----- ----- ~ ----- ---- ---- ---- ----
// Sim:  11000 ~ 11100 0011 1111 1100 0011
// Emu:  ----- ~ --0-- -1-- -0-- -0-- -1--
// Mism:       ~   *    *   *    *    *
// Monitor: EDT module channel 1 output.
```

The report format is similar to that of the K19 report. Here too, the gate is one the tool monitors for the purpose of analyzing where K22 errors first occur, so in addition to the simulation results, the report shows the values the tool emulated for the gate and whether there were any mismatches. Notice that the emulated values occur earlier in each cycle than with the K19 DRC. The reason is that the K22 DRC synchronizes emulation comparisons with the measure_sco statements in the shift procedure, whereas the K19 comparisons always occur at the end of each shift cycle.

Note

If you enter a constraint using a tool command, Tessent TestKompress does not apply it until the end of the test_setup procedure. You may thus see Xs up until the end of test setup; however, the tool does correctly apply the constraint before the start of load_unload or capture.

Example 8

The following example demonstrates the reporting of the timing exception and an internal pattern simulation value, and illustrates using the report_gates command to trace the transition of the pattern along the timing exception paths:

```
set_gate_report -timing_exceptions on
set_gate_report pattern_index 1
report_gates /ix21/UD1
report_gates /ix21/UD1

// /ix21/UD1 (36) DFF
```

```
// "S" I (0-0)(--) 6-
// "R" I (0-1)(--) 20-
// "C0" I (1-0)(--) 10-
// "D0" I (1-1)(To) 32-
// "OUT" O (1-1 [0])(In/Ef) 9-
// MASTER cell_id=2 chain=chain1 group=grp1 invert_data=FFFT
```

f

```
// /ix21/UD1 (9) BUF
// "I0" I (1-1)(Fr/Ef) 36-
// "OUT" O (1-1)(In) 18- 19-
```

f

```
// /ix21 (18) BUF
// "I0" I (1-1)(In) 9-
// Q O (1-1)(To) 28-/fdgd/A 29-/ix19/A
```

f

```
// /fdgd (28) NAND
// A I (1-1)(To) 18-/ix21/Q
// B I (1-0)(--) 1-/s
// Z O (0-1)(Ef) 37-/y[2]
```

f

```
// /y[2] (37) PO
// "I0" I (0-1)(Ef) 28-/fdgd/Z
// y[2] O (0-1)(Ef)
```

Example 9

The following example reports the static propagation time of gate */U1*:

set_gate_report delay_data static_propagation
report_gates /U1

```
//  /U1 (24)  XOR
//      AO     I  (5.4000, 5.5000)/(5.0000, 5.1000)  1-/PI1
//      A1     I  (2.4000, 2.4000)/(2.0000, 2.0000)  16-/U11/Y
//      Y      O  (2.4000, 2.0000)/(2.4000, 2.0000)  25-/U12/Y
```

The following example reports the static slack test of */U2*:

set_gate_report DELay_data STATIC_Slack
report_gates /U2

```
//  /U2  sff
//      D      I  (5.0000/2.0000)   /U21/Y
//      SI     I  (12.0000/12.0000) /U22/Q
//      SE     I  (NA/NA)  /SE
//      CLK    I  (8.0000/8.0000)   /CLK
//      RB     I  (STAT) /U23/Y
//      Q      O  (2.0000/3.0000)   /U24/SI /U25/A0 /U26/A1
```

The following example reports the local delay data for */U3*. The output in bold font highlights the timing information used by ATPG. Note, to report local delay data, you must set the gate level to primitive before executing the report_gates command as shown here:

set_gate_level primitive
set_gate_report delay local_delay_data
report_gates U3

```
//  /U3 (50)  BUF
//      "I0"   I  (-)  101-
//      Q      O  (-)  85-/U31/SI
//  /U3 (51)  INV
//      "I0"   I  (-)  101-
//      QB     O  (-)
//  /U3 (84)  MUX
//      SE     I  (-)  56-
//      D      I  (-)  78-/U32/Y
//      SI     I  (-)  48-/U34/Q
//      "OUT"  O  (-)  101-
//  /U3 (101)  DFF
//      "S"    I  (-)  69-
//      "R"    I  (-)  17-
//      CLK    I  (-)  54-
//      "D0"   I  (-)  84-
//      "OUT"  O  (-)  50-  51-
//
//   Sequential IOPATH delay starts through the fanin 3 of /U3/ (101):
//        1 condition delay(s) to /U3/Q (50)
//          condition delay: (1.0000, 2.0000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//      TimingCheck for data port at the fanin 2 of /U3/ (84):
//        SETUP - clock port at the fanin 3 of /U3/ (101)
//          condition delay: (0.7000, 0.7000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//        HOLD - clock port at the fanin 3 of /U3/ (101)
//          condition delay: (0.2000, 0.2000)
//            condition variable list:
//              V[0]=54
//            condition expression in post fix format: (V[0]) posedge
//
//      MASTER  cell_id=1  chain=c1  group=g1  invert_data=FFFF
//  Reported all gates in instance U3
```

Example 10

The following example reports gates with MCP (multicycle path) information, listing the minimum and maximum number of cycles. This example assumes that the following MCP information has been set from an SDC (Synopsys Design Constraint) file:

```
set_multicycle_path -through { get_pin [b1/I] }  -setup 2
```

set_gate_report -timing_exceptions on
report_gates /b1/ci2b cg2/TEN /cg2/i03

```
//  /b1/ci2b (47)  INV
//      A      I  (To/Th/1,2)  40-/b1/ci2a/Y
//      Y      O  (Ef/1,2)  49-/cg2/CLKEN

//  /cg2 (49)  NOR
//      CLKEN  I  (Ef/1,2)  47-/b1/ci2b/Y
//      TEN    I  (--)  3-/se
//      "OUT"  O  (Ef/1,2)  51-

//  /cg2/i03 (51)  TLA
//      "S"    I  (--)  9-
//      "R"    I  (--)  6-
//      "C0"   I  (--)  44-
//      "D0"   I  (Ef/1,2)  49-
//      "OUT"  O  (Ef/1,2)  11-
```

When the minimum and maximum number of cycles are the same and larger than 1, the command reports only one number. Single cycle paths and false paths do not report any number.

Example 11

The following example specifies that a subsequent report_gates command display the simulated values from the current simulation context:

> set_current_simulation_context stable_after_setup
> set_gate_report simulation_context
> report_gates {I1 I2}

```
/I1  primary_input
      I1     O  (0)  /AND2/A0
/I2  primary_input
      I2     O  (1)  /AND2/A1
```

Example 12

The following example enables the report_gates command to show the simulation values for the first F20 violation:

set_gate_report drc_pattern F20-1

Related Commands

report_gates
report_display_instances
set_gate_level

set_gzip_options

Context: unspecified, all contexts

Mode: all modes

Usage

set_gzip_options [-Path *gzip_path*] [-Fast | -Best | -*digit*]

Description

Specifies GNU gzip options to use with the GNU gzip command.

The set_gzip_options command specifies GNU gzip options the tool will use when compressing or decompressing files using the GNU gzip command. When file compression handling is enabled (as described under the set_file_compression command), the tool uses GNU gzip when processing files having a .gz extension.

The default speed and amount of compression corresponds to the gzip -4 option.

Arguments

- -Path *gzip_path*

  An optional literal that specifies the full path *gzip_path* to a gzip executable file. You need to provide this pathname only if gzip is not in your normal Linux search path. If you have specified a pathname with this option, you can restore the default behavior of using your Linux path to find gzip by issuing the command with the -Path gzip option.

The following switches all control the speed of compression:

- -Fast

  An optional switch that specifies to use the fastest compression method, which yields the least compression. This corresponds to the gzip -1 option.

- -Best

  An optional switch that specifies to use the slowest compression method, which yields the most compression. This corresponds to the gzip -9 option.

- -*digit*

  An optional switch that specifies an integer from 1 to 9 that the tool passes to gzip to control the rate of compression. You obtain the least amount of compression with -1, the greatest amount of compression with -9.

Example

The following example ensures file compression is enabled, sets gzip compression to the fastest method, then saves a file using the .gz file naming extension in order to activate gzip file compression handling:

```
set_file_compression on
set_gzip_options -fast
write_patterns fault.pat.gz
```

Related Commands

set_file_compression

write_patterns

set_icl_extraction_options

Context: all contexts

Mode: all modes

Usage

set_icl_extraction_options [-extract_learned_muxes {ON | OFf}]
[-extract_library_muxes {ON | OFf}]

Description

Customizes the behavior of ICL extraction.

For more information, see "ICL Extraction" in the *Tessent IJTAG User's Manual*.

Arguments

- -extract_learned_muxes {ON | OFf}

  An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from structures in the design that have been identified as muxes by the static learning process. The default for this switch is ON.

- -extract_library_muxes {ON | OFf}

  An optional switch and literal pair that specifies whether the tool creates ICL multiplexers from muxes that are described in the ATPG library. The default for this switch in ON.

Example

The following command switches off the recognition of multiplexer library cells in ICL extraction:

set_icl_extraction_options –extract_library_muxes off

Related Commands

get_icl_extraction_options
report_icl_extraction_options
set_static_learning

set_icl_scan_interface_ports

Context: dft, patterns -ijtag

Mode: setup

Usage

set_icl_scan_interface_ports **-name** ***name*** **-ports** ***port_objects***

Description

Sets the names of the ports in the specified scan interface previously added by add_icl_scan_interfaces that ICL extraction will create in the new ICL top module.

Arguments

- **-name** ***name***

  A required switch and value pair that specifies the name of scan interface for which the list of ports will be created in the new ICL top module.

- **-ports** ***port_objects***

  A required switch and value pair that specifies the names of the ports that will be created in the new ICL top module.

Example

The following example creates the ports tck, tdi, tdo, tms and trst for the ScanInterface TAP, and the ports tdo, tdrEn1, fromTdr1, ce, se and ue for the ScanInterface Internal in the new ICL top module:

set_current_design tap1

add_icl_scan_interfaces {TAP Internal}

set_icl_scan_interface_ports -name TAP -ports {tck tdi tdo tms trst}

set_icl_scan_interface_ports -name Internal -ports {tdo tdrEn1 fromTdr1 ce se ue}

The resulting ICL model will look like this:

```
Module tap1 {

   TCKPort         tck;
   ScanInPort      tdi;
   ScanOutPort     tdo       {  Source IRMux; }
   TMSPort         tms;
   TRSTPort        trst;
   ToSelectPort    tdrEn1    {  Source sel1;  }
   ScanInPort      fromTdr1;
   ToCaptureEnPort ce;
   ToShiftEnPort   se;
   ToUpdateEnPort  ue;

   ScanInterface TAP {Port tck; Port tdi; Port tdo; Port tms; Port trst;}
   ScanInterface Internal {Port tdo; Port tdrEn1; Port fromTdr1; Port ce;
Port se; Port ue;}
```

Related Commands

add_icl_scan_interfaces
delete_icl_scan_interfaces
get_icl_scan_interface_list
get_icl_scan_interface_port_list

set_iddq_checks

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

Mode: setup, analysis

Usage

set_iddq_checks [-NONe | {[-Int_float] [-EXt_float] [-Bus] [-WEAKBus] [-Pull] [-WIre] [-VOLTGain] [-VOLTLoss] [-Clock] [-WRite] [-Read]}
[-ATpg | -NO_ATpg] [-WArning | -ERror] [-LOOP_oscillation { OFf | REPort | REJect }]

Description

Specifies the restrictions and conditions to use when creating or selecting patterns for detecting IDDQ faults.

These restrictions only apply during the actual time of the IDDQ measurement; they are ignored at other times during a pattern.

To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

set_flattener_rule_handling fn1 error
set_flattener_rule_handling fn3 error

If you are using the tool to create Iddq patterns, a violation of any of the Iddq Check options you select causes the tool to reject an offending pattern.

During simulation, whenever violations of the restrictions occur, a message displays identifying the gate associated with the violation and the number of patterns in which the violations occurred. The handling of the violation can be either a warning or error.

If you select -Error, simulation terminates at the first occurrence of a violation. You can use the report_gates command to inspect the simulation values of all gates for patterns that violate the restrictions by first using the set_gate_report command with the Error_pattern option.

For more information on IDDQ, see "Creating an IDDQ Test Set" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- -NONe

  An optional switch that specifies to not perform any checks. This is the default.

- -Int_float — An optional switch that specifies to not allow a floating (Z) value on any ZVAL primitive gate that drives any Boolean logic. A ZVAL gate is placed between a source (Z producer) and a sink (nonZ consumer).

  A Z producer is any primitive that has a Z output. Z producers include top-level inputs, bidi/inouts, tristate drivers, and FETs/switches. A nonZ consumer is any gate that treats Z the same as X for all inputs. NonZ consumers include any Boolean gate. You can see the primitive truth tables for all DFT-supported primitives, in section "Supported Primitives" of the *Tessent Cell Library Manual*.

Note

A ZVAL gate is not placed between a Z producer and a PO or the PO half of a bidi because both of these have the ability to output a Z value. However, a ZVAL gate is placed between the PI half of the bidi and any internal _buf as typifies the input path of a bidi/inout IO pad. For those pads, if the output stage is disabled (Z out), and the bidi/inout PAD is driven externally by Z from the tester, and the IO pad has no modeled pullup/down, the ZVAL in front of the _buf input path can have a Z on its inputs; it fails the -int_float check.

Note

These “floating” checks only occur at buses. To ensure no input is floating due to an unconnected module or instance input, before the design is flattened, you should issue the following commands at the beginning of Setup:

```
set_flattener_rule_handling fn1 error
set_flattener_rule_handling fn3 error
```

- -EXt_float — An optional switch that prevents unidirectional tristateable outputs from being tristated (that is, prevents floating output-only pins). This switch is useful in unusual cases where the design cannot tolerate a Z out for IDDQ, but you should not use this switch unless absolutely necessary because it lowers IDDQ coverage. Use of this switch can also violate the IEEE 1149.1 TDO standard.
- -Bus — An optional switch that specifies to not allow contention conditions on bus gates.
- -WEAKBus — An optional switch that prevents contention conditions on weak-bus gates.
- -Pull — An optional switch that prevents contention conditions on pull gates.
- -WIre — An optional switch that specifies that all inputs of a wire gate must be set to the same value. There should be no contention on wires during IDDQ measurements because contention can raise the IDDQ current.
- -VOLTGain — An optional switch that specifies that a PMOS transistor must not be at a logic zero unless a zhold0 or zhold bus_keeper DFT library attribute is between the PMOS transistor and the BUS root. A compromised (Vss+) logic zero is pulled down to the Vss rail (zero volts).
- -VOLTLoss — An optional switch that specifies that a NMOS transistor must not pass a logic one unless a zhold1 or zhold bus_keeper DFT library attribute is between the NMOS

transistor and the BUS root. A compromised (Vdd-) logic one is pulled up to the Vdd rail (Vdd volts).

- -Clock — An optional switch that specifies to not allow clock pins to be on during the IDDQ measure.

Note

The advantage of the -Clock switch, during pattern generation, is that it prevents creation of Clock PO style patterns, one of the causes of multiple timeplates. The switch also ensures that when a clock is turned on, it is pulsed at the normal time.

- -WRite — An optional switch that specifies to not allow write control pins to be on during the IDDQ measure.
- -REad — An optional switch that specifies to not allow read control pins to be on during the IDDQ measure.
- -ATpg

  An optional switch that specifies to satisfy any IDDQ restrictions set using the add_iddq_exceptions command. This is the default.
- -NO_ATpg

  An optional switch that specifies to not enforce IDDQ restrictions set using the add_iddq_exceptions command. Patterns that do not satisfy restrictions are rejected by the fault simulator.
- -WArning

  An optional switch that treats violations of IDDQ checks as warnings. This is the default.
- -ERror

  An optional switch that treats violations of IDDQ checks as errors and stops the simulation process immediately.
- -LOOP_oscillation { OFf | REPort | REJect }

  An optional switch and keyword pair that checks for oscillating loops when the fault type is IDDQ. The “Report” option reports on oscillating simulation values at specific gates. The “Reject” option automatically rejects any patterns that cause oscillation and issues a warning. By default, the tool does not check for oscillating loops.

Example

The following example creates IDDQ patterns while checking that internal buses are not floating during an IDDQ measure and requires the deterministic ATPG to ensure that this is true:

```
set_system_mode analysis
set_fault_type iddq
set_iddq_checks -int_float
create_patterns
```

Related Commands

add_iddq_exceptions
analyze_restrictions
delete_iddq_exceptions
report_gates
report_iddq_exceptions
set_fault_type
set_gate_report
set_iddq_strobe

set_iddq_strobe

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

Mode: setup, analysis

Usage

set_iddq_strobe **-Label** | **-All**

Description

Specifies on which patterns (cycles) the tool will simulate IDDQ measurements.

The set_iddq_strobe command determines whether the tool simulates only existing IDDQ measure statements within an external pattern source, or if it should assume that every pattern (cycle) in the set has an IDDQ measure statement. The command performs slightly differently depending on which tool you are using. In either case, use the report_environment command to list the current setting. The following paragraphs describe how the set_iddq_strobe command operates for each tool.

The set_iddq_strobe command specifies which fault grading patterns will perform IDDQ measures during simulation.

Arguments

- **-Label**

  A switch that restricts IDDQ detection to those patterns which have the IDDQ measure statement. This is the default behavior upon invocation of the tool.

- **-All**

  A switch that allows the tool to use all patterns for IDDQ fault detection.

Example

The following example fault grades an external IDDQ pattern file and restricts IDDQ detection/measures to those patterns/test cycles which have the IDDQ measure statement:

```
set_system_mode analysis
read_patterns pat_file
set_fault_type iddq
set_iddq_strobe -label
add_faults -all
simulate_patterns
```

Related Commands

set_fault_type
set_iddq_checks
simulate_patterns

set_ijtag_instance_options

Context: dft patterns

Mode: setup analysis insertion

Usage

set_jtag_instance_options ***ijtag_instances*** [-use_in_dft_specification on | off | auto]

Description

Sets options on instances for which there is a matched ICL module.

Arguments

- ***ijtag_instance***

  A required literal that is the name of an Ijtag instance, a Tcl list of names of Ijtag instances, or a collection containing one or many Ijtag instances as reported by the get_instances or get_ijtag_instances commands.

  The instance objects must have an ICL module matched to them.

- -use_in_dft_specification on | off | auto

  An option value pair that defines the value of the -use_in_dft_specification option. When set to auto, the instances are included in the DftSpecification created by the create_dft_specification command when it is seen to not already be connected to the network. Specifying an on or off value bypasses the analysis and forces the inclusion or exclusion of the instances. See the description of the create_dft_specification command for more details about the usage of this option.

Example

The following example sets the -use_in_dft_specification option to off on all Ijtag instances below the instance blockA.

```
> set_ijtag_instance_options [get_ijtag_instances -below_instance blockA] \
                                -use_in_dft_specification off
```

Related Commands

get_ijtag_instances

get_ijtag_instance_option

report_ijtag_instances

set_ijtag_retargeting_options

Context: all contexts

Mode: all modes

Usage

set_ijtag_retargeting_options [-test_setup_network_end_state { keep | reset }]
[-compare_constant_capture_values { ON | OFf }]
[-compare_constant_data_out_ports { ON | OFf }]

Description

Enables you to configure different aspects of ijtag retargeting.

To use this command, you must have previously loaded and elaborated the top-level ICL module. Otherwise, the tool issues an error message.

Arguments

- -test_setup_network_end_state {keep | reset}

Note

This switch only applies to ijtag retargeting when an iCall is used in test_setup (see the set_test_setup_icall command for more details).

An optional switch and literal pair that specifies to switch off the reset and preserve the state of the ICL network after the last test_setup iCall. The default is to do the reset. This reset only affects the configuration of the scan paths and the selection of the scan interfaces. The reset does not trigger any activity on ports of type ResetPort or TRSTPort.

If you have a test mode where you are applying the load_unload data through the IJTAG network and do not want to reset the state of the ICL scan network after the last test_setup iCall, use the “set_ijtag_retargeting_options -test_setup_network_end_state keep” command to turn off the reset of the ICL network for those designs.

- -compare_constant_capture_values { ON | OFf }

  An optional switch and literal pair that turns on or off the automatic expected value generation for the scan registers.

  If turned on, the patterns will not include any extra scan loads. Instead, when a bit happens to be scan out in a given scan load and the value of this bit is known from the CaptureSource description in ICL, a compare value is automatically inferred for that bit at no extra cost. This feature is very useful for diagnosing issues with the network and should be kept on unless the ICL is incorrect.

- -compare_constant_data_out_ports { ON | OFf }

  An optional switch and literal pair that turns on or off the expected value generation for the primary DataOut ports. If turned on, the patterns will not include any extra vectors. An automatic compare value is inferred on top-level DataOutPorts if its value is known from its

described source in ICL. This feature is useful for confirming the correctness of the ICL but is not turned on by default to avoid increasing the number of used ports in a pattern file. For example, if you are exporting the pattern in SVF format, you typically would not want to have to compare output pins other than TDO.

Example

The following example loads the ICL and turns off the reset of the ICL network after the last test_setup PDL command.

```
SETUP> set_context patterns -ijtag -no_rtl
SETUP> read_icl ../data/chip.icl
SETUP> set_current_design

//  Note: Top design is 'chip'.

SETUP> set_ijtag_retargeting_options -test_setup_network_end_state keep
```

Related Commands

get_ijtag_retargeting_options

report_ijtag_retargeting_options

iApply

iRead

iWrite

set_insertion_options

Context: unspecified, all contexts

Mode: all modes

Usage

set_insertion_options [-auto_uniquify_edited_modules {on | off}] [-edited_module_prefix *edited_module_prefix*] [-net_uniquification_suffix *net_uniquification_suffix*] [-module_uniquification_suffix *module_uniquification_suffix*] [-instance_uniquification_suffix *instance_uniquification_suffix*] [-allow_assigns {on | off}] [-inserted_object_comment_label *label*]

Description

Specifies the values for the insertion options. The option settings affect the behavior of the insertion commands.

Arguments

- -auto_uniquify_edited_modules {on | off

  An optional switch and literal pair that tells Tessent Shell to automatically uniquify any module that needs to be edited and that has more than one instance in the current design.

  Note

  The current design is the design that was specified using the set_current_design command.

  During uniquification, the new module name is constructed by prefixing the original module name with the specified or default module_uniquification_suffix. The instantiation call of the module in its parent module is automatically changed to use the new module name. If the parent module is also non-unique, it is uniquified until a unique parent module is reached.

- -edited_module_prefix *edited_module_prefix*

  An optional switch and string pair that specifies a prefix to add to every edited module. A valid prefix contains numbers, letters, or underscores. Specifying a prefix is advisable when a module is used in multiple designs that are modified by Tessent Shell independently. For example, let's say you have a module called Counter that is used in CoreA and CoreB, and those two cores are processed in isolation, Adding a prefix that is a unique string to the module called Counter when editing the module in context of each core will prevent you from having different views of the same module with the same name if CoreA and CoreB are loaded together into a tool such as a simulator.

  The -edited_module_prefix defaults to a null string, which means that no module name prefixing is done by default.

- -net_uniquification_suffix *net_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a net when the net that is trying to be created already exists in the design. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the net name does not already exist. For example, specifying -net_uniquification_suffix of _TS# will append the string _TS# to the net where # is replaced by the first integer above 0, making the net name unique. If net A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new net with default name A and would use B_TS2 if it tried to create a new net with default name B.

  The net_uniquification_suffix defaults to "_ts#".

- -module_uniquification_suffix *module_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a module. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying -module_uniquification_suffix of _TS# will append the string _TS# to the module name where # is replaced by the first integer above 0, making the module name unique. If module A, B, and B_TS1 already exist in the design, Tessent Shell would use A_TS1 if it needed to create a new module with default name A and would use B_TS2 if it tried to create a new module with default name B.

  The module_uniquification_suffix defaults to "_#".

- -instance_uniquification_suffix *instance_uniquification_suffix*

  An optional switch and string pair that defines the string that Tessent Shell uses to uniquify a leaf instance which uses the create_instance command. The string must contain letters, numbers, or underscores and one and only one # symbol at the end identifying a counter that is auto-incremented starting from 1 until the module name does not already exist. For example, specifying -instance_uniquification_suffix of _TS# will append the string _TS# to the instance name where # is replaced by the first integer above 0, making the leaf instance name unique. If leaf instance A, B, and B_TS1 already exist in a module, Tessent Shell would use A_TS1 if it needed to create a new instance with default name A and would use B_TS2 if it tried to create a new instance with default name B.

- -allow_assigns <u>on</u> | off

  An optional switch and literal pair that specifies that assign statements are to be converted into Verilog primitive buffers when writing out the design. For example, creating a connection between nets that are also pins causes the creation of an assign statement. Those assign statements will show up as assign statements in the output netlist unless the -allow_assigns is set to off, in which case they will be replaced by buffer primitives.

- -inserted_object_comment_label *label* (-rtl context only)

  An optional switch and string pair that defines a label that the tool adds to comments appearing before and after each inserted object in the netlist. By default, the label is the null string “” which means no comments are inserted. For more information, refer to Example 2.

Example 1

The following example sets various insertion options that will affect the behavior of the insertion commands.

```
set_insertion_options -edited_module_prefix MyCoreName_ \
                      -module_unification_suffix _uniq#  -allow_assigns Off
```

Example 2

The following example sets the comment label for inserted objects to “myFlow”.

```
set_insertion_options -inserted_object_comment_label myFlow
```

A resulting netlist would then look like this:

```
module top(a,z, b);
   input a;
   output z;
   /* myFlow Ports Start */
   inout b;
   wire b;
   /* myFlow Ports End */
```

Example 3

The following example deactivates automatic commenting of inserted objects. This is the same as the default behavior if you do not specify the -inserted_object_comment_label argument at all.

```
set_insertion_options -inserted_object_comment_label { }
```

Related Commands

get_insertion_option

set_insert_test_logic_options

Context: dft -test_points

Mode: setup, analysis

Usage

set_insert_test_logic_options [**-inserted_object_prefix** ***prefix***]

Description

Specifies a prefix that overrides the default prefix “ts_” that is used when creating the test point logic.

Arguments

- **-inserted_object_prefix** ***prefix***

  A required switch and string pair that specifies a prefix that overrides “ts_” which is the default prefix. The *prefix* must consist of letters, numbers, or underscores.

Example

The following example shows how to override the default prefix for objects that are inserted into the netlist during test point insertion; the generated netlist will contain net and instance names that have the prefix “demo1_”:

```
set_insert_test_logic_options -inserted_object_prefix demo1_
insert_test_logic
write_design -output netlist.v -replace
```

Related Commands

get_insert_test_logic_options

set_internal_fault

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

Mode: setup

Usage

set_internal_fault **OFf** | **ON**

Description

Specifies whether the tool allows faults within or only on the boundary of library models.

The set_internal_fault command specifies whether the tool allows faults on internal nets of library models (*internal faulting*) or only on the library model boundary. The default upon invocation of the tool is to allow faults only on the library model boundary.

Note

This command is not valid for the bridge logic fault model (specified with the set_fault_type Bridge -Static command).

Arguments

- **OFf**

  A required literal that allows faults only on the boundary of the library models. This is the default.

- **ON**

  A required literal that allows faults on the internal nets of library models.

Example

The following Tessent FastScan example performs ATPG using a fault list that includes internal nets within the library cells. These internal nets will be shown as fault sites when the report_faults command is executed.

```
set_internal_fault on
set_system_mode analysis
create_patterns
report_faults -all
```

set_internal_name

Context: all contexts

Mode: setup

Usage

set_internal_name **OFf** | **ON**

Description

Specifies whether to delete or keep pin names of library internal pins containing no-fault attributes.

The set_internal_name command specifies whether to keep internal library pins with no-fault attributes. Normally, you should delete these names for memory and performance reasons. The default operation (OFF) upon invocation is to delete these names.

Arguments

- **OFf**

  A literal that deletes the lowest level pin names if they have the nofault attribute. This is the default.

- **ON**

  A literal that keeps the lowest level pin names, even if they have the nofault attribute.

Example

The following example deletes pin names with the nofault attribute.

```
set_internal_name off
```

set_io_mask

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

Mode: setup, analysis

Usage

set_io_mask OFf | ON

Description

Modifies the behavior of bidirectional pins so that their expected values will always be X during test cycles in which the primary input portion of the bidirectional pin is being forced.

Typically, when the tool forces stimulus on an bidirectional pin, it will expect to measure the same value on the corresponding primary output pin. This command lets you modify this behavior in cases where this is undesirable.

Arguments

- OFf

  An optional literal that disables the ability to mask the bidirectional pin output value. This is the default.

- ON

  An optional literal that enables the ability to mask the bidirectional pin output value.

Example

The following example causes the tool to mask the bidirectional pin output value.

```
set_io_mask on
```

set_latch_handling

Context: dft -scan

Mode: setup

Usage

set_latch_handling **None** | **Scan**

Description

Specifies whether the tool considers non-transparent latches for scan insertion while test logic is turned on.

The set_latch_handling command specifies whether the tool can consider non-transparent latches as candidates for scan insertion. If you determine that the tool is to consider non-transparent latches as candidates for scan insertion, you must turn on the appropriate set_test_logic command settings before the tool performs the rules checking process.

If you use the set_drc_handling command to set the D6 rule to error or warning, D6 checks the non-scannable latches for transparency. By default, if they are not transparent and you turned test logic insertion on, the tool does not consider the non-transparent latches for scan insertion. However, if you use the Scan argument with the set_latch_handling command, the tool will consider the non-transparent latches for scan insertion.

If you use the set_drc_handling command to ignore the D6 rule, then rules checking does not check the non-scannable latches for transparency and the tool will automatically consider all non-scannable latches, whose test logic you turned on, for scan insertion.

Arguments

- **None**

  A literal that specifies to give no consideration to non-transparent latches for scan insertion. This is the default upon invocation of the tool.

- **Scan**

  A literal that specifies to consider non-transparent latches for scan insertion when test logic is turned on.

Related Commands

set_drc_handling

set_test_logic

set_lbist_controller_options

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

Mode: setup, analysis

Usage

IP Creation Usage

set_lbist_controller_options {off | on} **-max_shift_cycles** {***integer1*** [*integer2*]}

-max_capture_cycles {***integer1*** [*integer2*]}

-max_pattern_count {***integer1*** [*integer2*]}

-capture_procedures {***procname*** [*integer%*]}…

Fault Simulation Usage

set_lbist_controller_options {off | on} **-capture_procedures** {***procname*** *integer%*}…

Description

Specifies global options to configure the logicBIST controller.

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- off | on

  An optional literal that specifies enabling or disabling logicBIST controller configuration. The default is on.

- **-max_shift_cycles** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum possible shift length the tool uses to synthesize the LogicBIST shift length counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the shift length counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum shift cycles.

  If you do not specify a value, then the maximum shift cycles value will be used for the hardware default mode.

- **-max_capture_cycles** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum possible capture width the tool uses to synthesize the LogicBIST capture width counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the capture width counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum capture width.

If you do not specify a value, then the maximum capture width will be used for the hardware default mode.

- **-max_pattern_count** {***integer1*** [*integer2*]}

  A required switch and integer pair or triplet that specifies the maximum number of LogicBIST patterns that will be applied when the tool synthesizes the LogicBIST pattern counter.

  An optional second argument, *integer2*, can be used to specify the value to be loaded in the pattern counter for the hardware default mode of operation. This value should be less than or equal to the ***integer1***, the maximum pattern count.

  If you do not specify a value, then the maximum pattern count will be used for the hardware default mode.

- **-capture_procedures** {***procname*** [*integer%*]}

  A required switch and repeatable string and optional integer percentage pair that specifies the Named Capture Procedure (NCP) name and its percentage of patterns for which the NCP should be applied.

  All the NCPs are repeated according to the specified percentage once every 256 patterns. The sum of all the specified percentages should add up to 100.

  Because the integer percentage may not be known at IP creation time, the integer percentage can be omitted and specified during fault simulation. At that time, you can program the percentage activity for each NCP without having to regenerate the IP. A percentage value of 0 means that the NCP will not be active for the pattern set.

Examples

The following example configures the logicBIST controller for a maximum scan chain length of 1000 and a maximum capture width of 8 cycles. The maximum number of patterns is set to 250,000 and number of patterns run in the hardware default mode is set as 5,000. There are 4 NCPs defined for LogicBIST test such that each of them is active for 25% of the patterns.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8

set_lbist_controller_options -max_pattern_count {250000 5000}

set_lbist_controller_options -capture_procedures \
        {clkseq1 25 clkseq2 25 clkseq3 25 clkseq4 25}
```

The following example is the same as the previous example except that the NCP percentage integer values have not been specified with the commands issued for IP creation. In this case, the NCP percentage integer values must be set by issuing the command with the desired values for fault simulation.

```
set_lbist_controller_options -max_shift_cycles 1000 -max_capture_cycles 8

set_lbist_controller_options -max_pattern_count {250000 5000}

set_lbist_controller_options -capture_procedures {clkseq1 clkseq2 clkseq3 clkseq4}
```

Related Commands

- report_lbist_configuration
- set_edt_options
- set_clock_controller_pins
- set_lbist_instances

set_lbist_instances

Context: dft -logic_bist

Mode: setup

Usage

set_lbist_instances [-controller_location *instance_path_name*]
[-single_chain_mode_logic_location *instance_path_name*]

Description

Specifies the instance in which the logicBIST controller or single chain mode logic is placed.

This command is used with the hybrid TK/LBIST flow. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

By default, the tool places the logicBIST controller and single chain mode logic in the top level of the design. Using this command, you can specify a different location for the logicBIST controller or single chain mode logic.

For example, you can use this command to place the logicBIST controller or single chain mode logic instance in a specific test logic block or in an always-ON block on a design with multiple power domains.

Arguments

- -controller_location *instance_path_name*

  An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the LogicBIST IP.

- -single_chain_mode_logic_location *instance_path_name*

  An optional switch and string pair that specifies the hierarchical instance pathname where the tool instantiates the single chain mode logic.

Examples

The following example instantiates the logicBIST controller in an always ON power domain instance */pwr_always/dft_logic*:

```
set_lbist_instances -controller_location /pwr_always/dft_logic
```

The following example instantiates the logicBIST controller and single chain mode logic in a dedicated test logic block instance */dft_test_logic_i*:

```
set_lbist_instances -controller_location /dft_test_logic_i \
                    -single_chain_mode_logic_location /dft_test_logic_i
```

Related Commands

set_edt_options

set_clock_controller_pins

set_lbist_pins

Context: dft -logic_bist

Mode: setup

Usage

set_lbist_pins
clock {***top_pin_name*** | **-**} [*internal_pin_name*] |
tck {***top_pin_name*** | **-**} [*internal_pin_name*] |
setup_shift_scan_in {***top_pin_name*** | **-**} [*internal_pin_name*] |
setup_shift_scan_out {***top_pin_name*** | **-**} [*internal_pin_name*] |
tap_instruction_decode {***top_pin_name*** | **-**} [*internal_pin_name*] |
test_logic_reset {***top_pin_name*** | **-**} [*internal_pin_name*] |
capture_dr {***top_pin_name*** | **-**} [*internal_pin_name*] |
shift_dr {***top_pin_name*** | **-**} [*internal_pin_name*] |
update_dr {***top_pin_name*** | **-**} [*internal_pin_name*] |
scan_en {{***top_pin_name*** |**-**} [*internal_pin_name* | **-**]}… |
xbounding_en ***pin_name***… |
control_point_en ***pin_name***… |
observe_point_en ***pin_name***… |
test_en ***pin_name***… |
mcp_bounding_en ***pin_name***… |

Description

Specifies the connection information for logicBIST controller pins.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

The pins you specify with this command are connected to the single top-level logicBIST controller. This command is in global context and not specific to the current EDT block.

The first argument is a literal string that indicates the pin being described in the command followed by one or more pin names.

If you issue this command multiple times with the same pin type, then the last pin or pins you specify are used. To specify multiple pin names corresponding to a pin type, specify all the names in a single issuance of the command.

If the signal to be connected to the logicBIST controller is internally generated and does not have a corresponding top-level pin, use a '-' literal in place of the ***top_pin_name*** argument.

Arguments

- **clock** {***top_pin_name*** | **-**} [*internal_pin_name*]

  A required literal and string pair that specifies the LogicBIST clock pin.

When a top-level tester controllable clock is available, the same pin can be used for both EDT and LogicBIST clocks. However, when the EDT clock is internally generated, a different free-running clock should be specified as the LogicBIST clock.

- **tck** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the JTAG TCK input and optionally the internal pin corresponding to the pad output.

- **setup_shift_scan_in** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the seeding register input. In a JTAG based design, this would typically be connected to the TDI pin.

- **setup_shift_scan_out** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the seeding register output. In a JTAG based design, this would typically be connected to the TDO pin through the DR-mux.

- **tap_instruction_decode** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the instruction decoder output that indicates when the TAP instruction that targets LogicBIST is loaded in the JTAG instruction register.

- **test_logic_reset** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the test_logic_reset output of the TAP controller that indicates when the TAP is in test-logic-reset state.

- **capture_dr** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the capture_dr output of the TAP controller that indicates when the TAP is in the capture-DR state.

- **shift_dr** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the shift_dr output of the TAP controller that indicates when the TAP is in the shift-DR state.

- **update_dr** {***top_pin_name*** | -} [*internal_pin_name*]

  A required literal and string pair that specifies the connection to the update_dr output of the TAP controller that indicates when the TAP is in the update-DR state.

- **scan_en** {{***top_pin_name*** |-} [*internal_pin_name* | -]}

  A required literal and string pair that specifies the scan enable pin and optionally the internal pin corresponding to the pad output. Multiple values can be specified if the design has multiple scan enables.

- xbounding_en ***pin_name***

  An optional literal and string pair that specifies the xbounding enable pins.

- control_point_en ***pin_name***

  An optional literal and string pair that specifies the control point enable pins. The pin names refer to the control_point_en pin created by Tessent Shell during test point insertion.

- observe_point_en ***pin_name***

  An optional literal and pin name that specifies the observe point enable pins. The pin names refer to the observe_point_en pin created by Tessent Shell during test point insertion.

- test_en ***pin_name***

  An optional literal and string pair that specifies the test enable pins. The pin names refer to the test_en pin created during scan insertion.

- mcp_bounding_en ***pin_name***

  An optional literal and string pair that specifies the MCP bounding enable pins. The pin names refer to the mcp_bounding_en pin created during X-bounding to bound false and multicycle paths.

- ***top_pin_name* | -**

  A required string or literal that specifies the pin name to be connected to the LogicBIST controller. Choose from the following:

  top_pin_name — A string that specifies the name of the pin to assign to the specific signal.

  - — A literal used for internally-generated pins and do not have a corresponding top-level pin.

- *internal_pin_name*

  An optional string that specifies an internal pin name.

Example 1

This example specifies the connections between design pins and logicBIST controller pins, including the TAP interface. A dash is used as the first argument for the TAP generated control signals, since there is no corresponding top-level pin for these signals.

```
set_lbist_pins clock refclk

set_lbist_pins scan_en scan_en

set_lbist_pins xbounding_en lbist_en

set_lbist_pins setup_shift_scan_in tdi

set_lbist_pins setup_shift_scan_out { tdo jtag/scanCfgReg_so }

set_lbist_pins shift_dr { - jtag/shift_dr }

set_lbist_pins capture_dr { - jtag/capture_dr }

set_lbist_pins update_dr { - jtag/update_dr }

set_lbist_pins test_logic_reset { - jtag/tlr }

set_lbist_pins tap_instruction_decode { - jtag/scanCfgReg_en }
```

Example 2

In this example, the design pin “lbist_en” is used as the enable signal for x-bounding, control points, and observe points. Note that the same pin is specified for all three pin types. The logicBIST controller will have only a single enable register named “x_bounding_en.”

set_lbist_pins xbounding_en lbist_en

set_lbist_pins control_point_en lbist_en

set_lbist_pins observe_point_en lbist_en

Example 3

This example illustrates how to specify multiple scan enable signals without pads. Since the scan enable pin names should be specified in pairs, a dash is used for each of the non-existent internal pin names.

set_lbist_pins scan_en {scan_en - wrp_scan_en -}

Related Commands

set_lbist_controller_options

report_lbist_pins

set_edt_options

set_lbist_power_controller_options

Context: dft -logic_bist

Mode: setup

Usage

set_lbist_power_controller_options [**shift** {**none** | **disabled** | **enabled**}
[-min_switching_threshold_percentage ***sw_percentage***]]

Description

Specifies creation of the low-power shift controller for LogicBIST.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

This command is applicable to the current LogicBIST block.

When you specify this command during IP creation, the tool creates the low-power shift controller for LogicBIST. This command is carried forward in the generated fault simulation dofile to indicate whether the current EDT block has low-power shift logic synthesized (automatically added to the fault simulation dofile).

Arguments

- **shift none | disabled | enabled**

  A literal pair that indicates the configuration parameters for controlling shift power.

  Choose one of the following:

  - **none** — Indicates that the low-power controller is not synthesized.
  - **disabled** — Forces a LogicBIST run to bypass the low power controller.
  - **enabled** — Enables the low-power controller.

  When you specify either 'enabled' or 'disabled', a low-power controller with the specified switching threshold percentage is synthesized.

- -min_switching_threshold_percentage ***sw_percentage***

  An optional switch and integer that specifies the switching threshold value.

  Valid integer values are between 1 and 50. The default value is 15 percent.

  The low-power controller will be synthesized that can support a range of actual switching percentages around the specified value. During fault simulation, any switching_threshold higher than this minimum can be specified.

Example

This example shows three EDT/LogicBIST blocks, two of which are to be synthesized with both EDT and LogicBIST low-power logic, while the third block does not have low power shift logic.

add_edt_block lpblk1

set_edt_power_controller shift enabled -min_switching_threshold 20

set_lbist_power_controller_options shift enabled

add_edt_block lpblk2

set_edt_power_controller shift enabled -min_switching_threshold 8

set_lbist_power_controller_options shift enabled

add_edt_block rgblk

Related Commands

set_power_control

set_edt_power_controller

set_learn_report

Context: all contexts

Mode: setup, analysis

Prerequisites: The current design must be set.

Usage

set_learn_report OFf | ON

Description

Specifies whether the report_gates command can display the learned behavior for a specific gate.

The set_learn_report command specifies whether the report_gates command should include the information that the tool collects during the static learning process. The application automatically performs the static learning process immediately after it flattens the simulation model, which happens when you leave the Setup mode or issue the create_flat_model command. The static learning process provides general information on the design that the tool can then use in speeding up the ATPG process (such as values that are impossible on other gates if the selected gate is at a specific value.)

Once you enable access to the static learned information with the set_learn_report command, you can specify for the tool to display the learned information on a selected gate by using the report_gates command.

While you can also access the learned information with the report_gates command by using the -Type option, this method displays the information for all the gates of the specified gate type. When you enable access with the set_learn_report command, the tool automatically displays the learned information with all command options. Therefore, you can restrict the report to the learned information on an individual object.

Arguments

- OFf

  An optional literal that disables access to the learned behavior information. This is the default.

- ON

  An optional literal that enables access to the learned behavior information.

Example

The following example enables access to the learned behavior and then accesses that information:

set_learn_report on
report_gates 28

```
/MX3/OR1 (28)  OR
    IO     I  20-/MX3/AN2/OUT
    I1     I  24-/MX3/AN1/OUT
    OUT    O  37-/OUT0
    Learned behavior:  MUX(9,13,17)
```

Related Commands

report_gates

set_lfsr_seed

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

set_lfsr_seed ***name seed [***-Binary | -Hexadecimal***]***

Description

Sets the initial value of a PRPG or a MISR.

The set_lfsr_seed command initializes the value of a PRPG or MISR with a seed value. The seed value must be specified by a hexadecimal or binary number greater than zero.

Arguments

- ***name***

  A required string that specifies the name of the PRPG or MISR.

- ***seed***

  A required number, greater than zero, that specifies the initial state of the LFSR.

- -Binary

  An optional switch that specifies to output the value of the *seed* argument in binary format. By default, the seed is output in hexadecimal.

- -Hexadecimal

  An optional switch that specifies to output the value of the *seed* argument in hexadecimal format. This is the default.

Example

set_lfsr_seed prpg 5

Related Commands

add_lfsr_taps
add_lfsrs
delete_lfsrs

set_lfsrs

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup

Usage

set_lfsrs [-Both | -Serial | -Parallel] [-Out | -In | -Ca]

Description

Changes the *shift_type* and *tap_type* default setting for the add_lfsrs and add_lfsr_taps commands.

The set_lfsrs command controls the default setting for the *shift_type* and *tap_type* switches. You specify the LFSR's shift technique by using one of the following *shift_type* switches: -Both, -Serial, or -Parallel. You specify the placement of the exclusive-OR taps by using one of the following *tap_type* switches: -Out or -In; or you specify the type of LFSR as celluar automata using the *tap_type* switch -*Ca*. When you change one or both of the default settings, all future add_lfsrs and add_lfsr_taps commands use the new default.

Arguments

The following lists the three *shift_type* switches of which you can choose only one:

- **-Both**

  An optional switch specifying that the LFSR shifts both serially and in parallel. This is the default behavior upon invocation of the tool.

- **-Serial**

  An optional switch specifying that a serial shift LFSR shifts a number of times equal to the length of the longest scan chain for each scan pattern.

- **-Parallel**

  An optional switch specifying that a parallel shift LFSR shifts once for each scan pattern.

The following lists the three *tap_type* switches of which you can only choose one:

- **-Out**

  An optional switch that places the exclusive-or taps outside the register path. This is the default upon invocation of the tool.

- **-In**

  An optional switch specifying that the exclusive-or taps are placed outside the register path.

- **-Ca**

  An optional switch specifying that the default PRPG type is cellular automata.

Example

The following example changes the default *shift_type* setting to Serial and the default *tap_type* switch to In:

```
set_lfsrs -serial -in
add_lfsrs lfsr1 prpg 5 13
add_lfsrs lfsr2 prpg 5 11
add_lfsr_taps lfsr1 2 3
add_lfsr_taps lfsr2 3 4
set_system_mode analysis
```

Related Commands

add_lfsrs
add_lfsr_taps
delete_lfsrs
delete_lfsr_taps
report_lfsrs

set_license_queue_timeout

Context: all contexts

Mode: all modes

Usage

set_license_queue_timeout { *value* | unlimited | no_queue }

Description

Specifies the amount of time Tessent Shell will queue for a required license if it is not immediately available.

This command configures the time the tool will wait for getting a required license (for example, during set_context). You can specify the same value during tool startup with the command line switch -license_wait (for example, tessent -license_wait).

If the license is not acquired during the specified time, an error is reported and the operation that requested the license is aborted. You can interrupt the license queuing with Ctrl-C. This will result in an error and the operation that requested the license(s) will be aborted.

Arguments

- *value*

  A positive integer that specifies the queuing time in minutes.

- unlimited

  An optional string that specifies that there is no time limit and the tool will wait until a license is available. This is the default behavior.

- no_queue

  An optional string that specifies that the tool will not queue for a license and will issue an error if it is not available immediately.

Example

The following example sets the license queue timeout value to 3 minutes, then displays the timeout value that was set:

```
SETUP> set_license_queue_timeout 3
SETUP> get_license_queue_timeout
```

```
3
```

Related Commands

get_license_queue_timeout

tessent

set_list_file

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

Mode: setup, analysis

Usage

set_list_file [*filename*] [-Replace]

Description

Specifies the name of the list file into which the tool places the pins' logic values during simulation.

The set_list_file command specifies the file in which the tool places the simulation values for the pins which you previously identified with the add_lists command. The default behavior is for the tool to display the simulation values for the pins on standard output.

You can display the list of reported pins by using the report_lists command.

Arguments

- *filename*

  A string that specifies the name of the file in which the tool places the logic values of pins during simulation. If you do not provide a *filename*, the default is standard output.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Example

The following Tessent FastScan example creates a file to store simulation values that are being reported:

```
set_system_mode analysis
add_lists  /i_1006/o  /i_1007/o
set_list_file listfile
create_patterns
```

Related Commands

add_lists
delete_lists
report_lists

set_lockup_cell

Context: dft -scan

Mode: setup, analysis

Prerequisites: Lockup cell model(s) must be specified with the add_cell_models command and an inverter model must be specified with the add_cell_models command or cell_type model attribute.

Usage

set_lockup_cell {**OFf** | **ON**} [-First_clock | -SEcond_clock] [-Type {DLat | DFf}]
[-CApture_edge_at_scan_chain_input {LE | TE | ANY}]
[-CHange_edge_at_scan_chain_output {LE | TE | ANY | OFF} [{ALL | WRApper}]]

Description

Enables the automatic insertion of lockup cells.

You can insert the lockup cells at the beginning/end of each scan chain or between all scan cells in different clock/edge domains. The lockup cells are inserted only on the scan path and do not interfere with the functional operation of the design.

Lockup cells are inserted at each transition of the different clock domains and at the transition of active-high to active-low edge domains. The edge domain of a scan cell is determined by the combined effect of the following parameters:

- The off-state value of the feeding top-level clock, which is defined by the add_clocks command.
- The overall inversions along the clock path, from the clock pin of the scan cell to the top level clock pin.
- The inversions on the clock path within the scan cell library model, which is defined by using the clock input attribute statement and/or _inv primitives in the library model definition.

Lockup cell and inverter models must be defined in the DFT library before they can be used in lockup cell insertion. Also, a lockup cell model of type DLAT or DFF must be specified using the add_cell_models command. You must use the -Active switch to specify the overall signal inversion on the enable pin of the latch (or the clock pin of the flip-flop) inside the library model. The tool does not examine or simulate the implementation of the lockup model to determine if it is an active high or active low cell. In the following example, the overall inversion on the enable pin is active high. For more information, see the add_cell_models command.

```
model lockup (D, CLK, Q, QB) (
     cell_type = DFF CLK D;
     input (D) ()
     input (CLK) (clock = fall_edge;)
     intern(CLKn) (primitive = _inv (CLK, CLKn);)
     output(Q, QB) (primitive = _dff(, , CLKn, D, Q, QB);)
```

)

If a cell order file is used with the insert_test_logic command, lockup cells are inserted only at the locations specified in the cell order file. If you are using a DLAT model as a lockup cell, you must specify the model type using the add_cell_model command. If you are using a DFF model as a lockup cell, you must specify the model type using the set_lockup_cell command because it is the only way to specify the type of library model to use for lockup cells. Using the set_lockup_cell command also turns on the reporting of those locations that require lockup cells in theory but are not covered in the cell order file. If no lockup cell location is specified in the cell order file, the tool inserts lockup cells automatically at the required locations.

For more information, see the insert_test_logic command.

Table 7-42 illustrates how the tool inserts lockup cells at different clock/edge domains.

Table 7-42. Lockup Cell(s) Used Between Different Clock/Edge Transitions

Clock/Edge Transition	Lockup Cell Models Defined	Lockup Cell Inserted
$c1^+$ --> $c2^+$	lat^+	$c1^+$ --> (lat^+ + inv) --> $c2^+$
$c1^+$ --> $c2^+$	lat^-	$c1^+$ --> (lat^-) --> $c2^+$
$c1^+$ --> $c2^+$	lat^+, lat^-	$c1^+$ --> (lat^-) --> $c2^+$
$c1^-$ --> $c2^-$	lat^+	$c1^-$ --> (lat^+) --> $c2^-$
$c1^-$ --> $c2^-$	lat^-	$c1^-$ --> (lat^- + inv) --> $c2^-$
$c1^+$ --> $c2^-$	lat^+, lat^-	$c1^+$ --> (lat^-) --> $c2^-$
$c1^-$ --> $c2^+$	lat^+, lat^-	$c1^-$ --> $c2^+$ (no lockup)

The first column lists a clock/edge transition where an active high-clock 1 to an active-low clock 2 is shown as $c1^+$ --> $c2^-$.

The second column lists the lockup cell models defined in the DFT library where lat^+ is an active-high model and lat^- is an active-low model.

The third column lists the transitions after lockup cell insertion where the lockup cell inserted is listed in the parenthesis. A lockup cell labeled as (lat^+ + inv) in the table indicates that a lockup model is inserted in the scan path and an inverter is inserted on the clock line of the lockup cell to provide a half cycle delay. In Table 7-42, it is assumed that the clock signal is tapped from the clock input of the leading flop. The inverter is used when an active low model is needed but not defined.

The tool can also insert lockup cells at the non-transition locations in a chain, namely, at the beginning and at the end of the chain. The -Capture_edge_at_scan_chain_input switch provides the ability to control the time when the first cell of the scan chain captures a value. To guarantee a capture edge (specified with LE or TE argument) at the first cell in chain, the tool inserts a lockup cell before the first cell in the chain (closer to the scan chain input) if the first cell has the opposite capture edge. If the ANY argument is specified, the tool always inserts a lockup cell at the beginning of the chain with the opposite capture edge of the first cell. The tool uses the clock signal of the first cell with the opposite clock edge when a DLAT model is used and with

the same clock edge when a DFF model is used for a lockup cell. Note that the capture and the change edges for a DLAT model are the opposite, whereas for a DFF model, they are the same. For example, for an active high DLAT model, the capture edge is trailing (TE) and the change edge is leading (LE).

Similarly, for a leading edge DFF model, the capture and change edges are both leading (LE).

The -Change_edge_at_scan_chain_output switch provides the ability to control the change time at the output of the last scan cell in the chain. To guarantee a change edge (specified with LE or TE argument) at the last cell in chain, the tool inserts a lockup cell after the last cell in the chain (closer to the scan chain output) if the last cell has the opposite change edge. If the ANY argument is specified, the tool always inserts a lockup cell at the end of the chain with the opposite change edge of the last cell. The tool uses the clock signal of the last cell with the opposite clock edge regardless of the latch model being used (DLAT or DFF) for a lockup cell.

Lockup cells are inserted based on the change edge of the source cell clock and the capture edge of the destination clock. For a lockup cell that will be inserted at the beginning of a scan chain, the source cell does not exist and therefore the capture edge of the destination cell (first cell in chain) is specified. Similarly, for lockup cell that will be inserted at the end of a scan chain, the destination cell does not exist and the change edge of the source cell (last cell in chain) is specified. The lockup cells between two cells within the scan chain are inserted automatically by the tool since both source and destination cells are available.

For more information on change edge and capture edge, refer to "Lockups Between Decompressor and Scan Chain Inputs" in the *Tessent TestKompress User's Manual*.

For more information on inserting lockup cells, refer to "Merging Chains with Different Shift Clocks" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **OFf | ON**

  A required literal that determines whether lockup cells are inserted. By default, lockup cells are inserted.

- -First_clock | -SEcond_clock

  An optional switch that determines which clock signal the lockup cells use. Options include:

  -First_clock — clock signal is tapped from the clock input of the scan cell closer to the scan chain input. By default, the first clock is used.

  -SEcond_clock — clock signal is tapped from the clock input of the scan cell closer to the scan chain output.

- -Type {DLat | DFf}

  Optional switch and literal pair that specifies the type of model used for a lockup cell. Options include:

  DLat — Specifies D latch with two input pins (enable and data).

  DFf — Specifies a D flip-flop with two input pins (clock and data).

- -CApture_edge_at_scan_chain_input {**LE | TE | ANY**}

  Optional switch and literal pair that determines whether a lockup cell is inserted between the first scan cell and the scan input pin of scan chains. This switch allows you to control the capture edge used by the first scan cell. Options include:

  LE — lockup cell is inserted to ensure that the first scan cell captures data on the leading edge.

  TE — lockup cell is inserted to ensure that the first scan cell captures data on the trailing edge.

  ANY — lockup cell is inserted independently from the capture edge of the first scan cell.

  If a DFF model is to be used as a lockup cell, the clock signal obtained from the first scan cell in chain is inverted before connecting it to the clock input of the lockup cell. If a DLAT model is used, on the other hand, the clock signal is not inverted.

- -CHange_edge_at_scan_chain_output {**LE | TE | ANY | OFF**}

  Optional switch and literal that determines whether a lockup cell is inserted between the last scan cell and the output pin of scan chains. This switch allows you to control the capture edge used by the last scan cell. Options include:

  LE — lockup cell is inserted to ensure that the last scan cell can update its output on the leading edge.

  TE — lockup cell is inserted to ensure that the last scan cell can update its output on the trailing edge.

  ANY — lockup cell is inserted independently from the change edge of the last scan cell.

  OFF — lockup cells are not inserted at the end of any chains (wrapper or core).

  The clock signal obtained from the last scan cell in chain is always inverted before connecting it to the clock input of the lockup cell whether a DFF model or a DLAT model is used.

- {ALL | WRApper}

  Optional switch that specifies the insertion of lockup cells at the end of all chains or of only wrapper chains.

  ALL — a lockup cell is inserted at the end of all chains.

  WRApper — a lockup cell is inserted at the end of all wrapper chains.

Example 1

The following example defines two different clocks, specifies a flop model and inverter model to use for lockup cells, enables lockup cell insertion, and performs the insertions. The *-Clock Merge* option combines the scan cells associated with each of the specified clock groups into a scan chain when the test logic is inserted.

```
add_clocks 0 clk1
add_clocks 1 clk2
add_cell_model dff04 -type dff clk data
add_cell_model inv -type inv
set_lockup_cell on -type dff
insert_test_logic -scan on -clock merge
```

In this example, the tool creates one scan chain and inserts lockup cells between the two clock domains.

Example 2

The following example defines a latch model and inverter model to use for lockup cells, turns on the insertion of lockup cells to ensure that the first scan cell captures data on the leading edge, and inserts the test logic.

```
add_cell_model dlat1a -type dlat enable data
add_cell_model inv -type inv
set_lockup_cell on -capture_edge_at_scan_chain_input LE -type dlat
insert_test_logic -scan on
```

Example 3

The following example inserts lockup cells at the end of all wrapper chains with ANY capture edge and at the end of core chains with a LE capture edge. The following commands must be executed in the order shown because the effect of the switches is cumulative. Note that the second command overrides the capture edge constraint for wrapper chains only which leaves the LE capture constraint to apply to core chains only.

```
set_lockup_cell on –change_edge_at_scan_chain_output le all
set_lockup_cell on –change_edge_at_scan_chain_output any wrapper
```

Related Commands

add_cell_models
add_tied_signals
insert_test_logic

set_logfile_handling

Context: unspecified, all contexts

Mode: all modes

Usage

set_logfile_handling [-RESume | {*filename* [-Replace | -Append]}]

Description

Specifies for the tool to direct the transcript information to a file.

The set_logfile_handling command causes the tool to write the transcript information, which includes the commands and the corresponding output (if any), into the file you specify. You can execute the set_logfile_handling command at any time, as many times as you need.

In the logfile, the **command** keyword (or **sub-command** when a tool command is embedded in a Tcl command) precedes all commands that the tool executes. You can easily search for the executed commands, generate a separate dofile containing those commands, and then execute the dofile, thereby rerunning those commands within the tool.

When you set the logfile handling, the tool still writes the same information to the session transcript window in addition to the logfile. However, you can disable the writing of the information to the transcript window with the set_screen_display command.

If you want to stop writing to a logfile, issue the set_logfile_handling command with no options, which closes the appropriate file.

Arguments

- -Resume

  An optional switch that specifies for the tool to resume writing transcript output to the logfile specified at invocation with the -Logfile invocation switch. You do not need to include the name of the invocation logfile with this switch; the tool will remember the name.

- *filename*

  A string that specifies the name of the file to which you want the tool to write the transcript output. This string can be a full pathname or a leafname. If you only specify a leafname, the tool creates the file in the directory from which you invoked the tool.

  If you do not specify a *filename*, the tool discontinues writing to the logfile and closes it.

- -Replace

  An optional switch that forces the tool to overwrite the file if a file by that name already exists.

- -Append

  An optional switch that causes the tool to begin writing the transcript at the end of the specified file.

Related Commands

set_screen_display
set_transcript_style
tessent

set_logical_design_libraries

Context: all contexts

Mode: setup

Usage

set_logical_design_libraries **-logical_library_list** ***logical_library_list***
[-default_library *default_library_name*] [-force]

Description

Defines the set of available logical libraries that can be used by the read_verilog and read_vhdl -in_library option.

You must use this command before any design modules have been read in unless you specify the -force switch. If you specify -force, all design data that was read before is lost and must be re-read or recreated.

In the no_rtl context, this command is not allowed; a single library called “work” is used.

As you can see in the example, you can create a Tcl list of library names and use it as the *logical_library_list* value. You can sort this Tcl list from the bottom up to order the VHDL libraries into an order such that the libraries that depend on other libraries are defined after those they depend on. The default library is the last item in the list unless the -default_library is used to specify a different one.

Arguments

- **-logical_library_list** ***logical_library_list***

  A required switch and string pair that specifies the list of logical libraries.

- -default_library *default_library_name*

  An optional switch and string pair that specifies the library name to be used as the default library. If this switch is not specified, the last library name in the logical_library_list is used as the default library.

  Except for the get_modules command, all commands that accept a module name as one of their options look for the module name inside the default library. For example, to create an instance of module M1 from library L1, you use the following command:

  ```
  create_instance instance_name -of_module [get_module M1 -in_library L1]
  ```

  Using -of_module M1 creates an instance of module M1 from the default library.

- -force

  An optional switch that suppresses the error message normally displayed when this command is executed after modules have already been read in or created. When the -force option is used, all pre-existing modules are deleted.

Example

In this example, a Tcl list is defined with all logical library names and is ordered with the libraries that depend on other libraries defined after those they depend on. A Tcl array called vhdl_files is used to specify which files goes in which libraries. A simple design-independent script at the bottom is used to load those VHDL files into the right libraries. A similar technique can be used for loading Verilog into multiple design libraries.

```
set vhdl_dir design/vhdl/src

set lib_list [list utility lib1 work]

set vhdl_files(utility) [list \
        ${vhdl_dir}/f1.vhd\
        ${vhdl_dir}/f2.vhd\
]

set vhdl_files(lib1) [list \
        ${vhdl_dir}/f3.vhd\
        ${vhdl_dir}/f4.vhd\
]

set vhdl_files(work) [list \
        ${vhdl_dir}/f5.vhd\
        ${vhdl_dir}/f6.vhd\
]

set_logical_design_libraries -logical_library_list $lib_list

foreach lib $lib_list {
        if {[array names vhdl_files -exact $lib] == $lib} {
            read_vhdl $vhdl_files($lib) -in_library $lib
        }
}
```

Related Commands

get_logical_library_list
read_verilog
read_vhdl
set_current_design

set_loop_handling

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

Mode: setup

Usage

set_loop_handling [{Tiex [-Duplication {ON | OFf}]} | Simulation]
[-MAX_Stored_loops { *number* | UNLIMITED }]

Description

Specifies how the tool handles feedback networks.

The set_loop_handling command lets you perform DRC simulation of circuits containing combinational feedback networks.

The set_loop_handling command specifies the tool's loop handling behavior by either inserting a TIE-X gate or by stabilizing the loop values through an iterative simulation process.

By using the Tiex setting, you have the option to use gate duplication to reduce the impact that a TIE-X gate places on the circuit to break combinational loops. By default, this duplication switch is off.

For an iterative simulation, the tool inserts FB_BUF gates to break the combinational loops. It also optimizes the number of iterations performed to both stabilize the circuit and minimize the impact of the iterative process on performance and memory usage.

For another look at combinational feedback loops, refer to "Feedback Loops" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- Tiex

  A literal that specifies that TIE-X gates are used to break combinational loops.

- Simulation

  A literal that specifies for the tool to use a simulation process to stabilize values in the loop. This option gives more accurate simulation results than other options. This is the default.

- -Duplication ON | OFf

  An optional switch and literal pair that specify whether the tool can insert duplicate gates to reduce the impact of the gates that the tool places to break combinational loops. The literal choices are as follows:

  - ON — An optional literal that specifies for the circuit learning process to generate duplicate gates within any identified feedback paths.
  - OFf — An optional literal that specifies for the circuit learning process to not generate duplicate gates within any identified feedback paths. This is the default upon invocation.

- -MAX_Stored_loops { *number* | UNLIMITED }

  An optional switch and integer or keyword pair that specifies the maximum number of loops that can be stored for reporting (using the report_loops command). By default, the tool stores up to1000 loops. Use this switch if you want to store a specific number or all loops for reporting after the flattening process.

Example 1

The following Tessent FastScan example inserts a TIE-X gate to break any identified combinational or sequential asynchronous loop, then performs ATPG:

```
set_loop_handling tiex
set_system_mode analysis
create_patterns
```

Example 2

The following example shows how the tool indicates a design with more than 1000 combinational loops during the flattening process, and how to store all loops and write the loops to file loops.txt.

```
flatten_model

//  Note: Number of combinational loops exceeds storage limit (1000).

//      Use 'set_loop_handling -max_stored_loops' to modify the number of loops stored and reported
//       by 'report_loops'.

//  Flattening process completed,  design_cells=28234  leaf_cells=1122 8 library_primitives=22323
//  sim_gates=45400  PIs=110  POs=103  CPU time=93.00 sec.

//  ---------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ---------------------------------------------------------------------------
//  Equivalent gates=54   classes=0   CPU time=9.00 sec.

set_loop_handling  -max_stored_loops unlimited

//  Warning: Flattened model deleted.

flatten_model

//  Flattening process completed,  design_cells=28234  leaf_cells=1122 8 library_primitives=22323
// sim_gates=45400  PIs=110  POs=103  CPU time=93.00 sec.

//  ---------------------------------------------------------------------------
//  Begin circuit learning analyses.
//  ---------------------------------------------------------------------------
//  Equivalent gates=54   classes=0   CPU time=9.00 sec.

report loops > loops.txt
```

Related Commands

set_tla_loop_handling

report_loops

set_lpct_condition_bits

Context: dft -edt

Mode: setup

Usage

set_lpct_condition_bits {**-Condition** {**Reset** | **Scan_enable**}
[*pin_name* –on_controller_module] {**-From** ***connection_name*** | **-No_connection**}

Description

Specifies the condition scan cells (condition bits) in the design core to connect to the Type 3 low pin count test (LPCT) controller.

The condition scan cells (condition bits) must already exist in the design core. If no condition bits are specified, the LPCT controller fully controls the reset and scan_enable signals. When LPCT condition bits are specified, the appropriate control bits are added to the test configuration data register in the LPCT controller. For each scan enable condition and each reset condition, one bit is added to the test configuration data register.

During ATPG, the scan cell specified using set_lpct_condition_bits *-from connection_name* is justified to the correct value to ensure the correct value of reset and scan enable.

This command is used in the IP Creation phase; it can only be used to create the Type 3 LPCT controller. For more information, see the Type 3 Controller in "LPCT Controller Types" in the *TestKompress User's Manual*.

Arguments

- **-Condition** {**Reset** | **Scan_enable}**

  Required switch and literal pair that specifies the condition bit type and the condition scan cell pin pathname. Literal options include:

  Reset — Condition bit for controlling design reset during capture.

  Scan_enable — Condition bit for controlling design scan enable during capture.

- *pin_name* –on_controller_module

  Optional string and switch pair that specifies the pin name in the controller RTL for that particular input; this does not specify the connection.

- **-From** ***connection_name*** | **-No_connection**

  Optional switch and string or switch that specifies the pin corresponding to the condition bits. This argument specifies the input connection to the LPCT controller. The *–no_connection* switch removes a previously specified condition.

Example

The following example defines condition bits in the design to control the reset and scan_enable signals for the LPCT controller:

```
// Condition scan cell outputs (will be connected to LPCT logic)
set_lpct_condition_bits –condition reset -from c1/Q
set_lpct_condition_bits –condition scan_en -from c2/Q
```

Related Commands

report_lpct_configuration
report_lpct_pins
set_lpct_controller
set_lpct_controller
set_lpct_instances
set_lpct_pins

set_lpct_controller

Context: dft -edt, patterns -scan (EDT On), patterns -scan_diagnosis

Mode: setup

Usage

set_lpct_controller **OFF** |**On** [-MAX_SHift_cycles shift_cycles]
[-MAX_CApture_cycles capture_cycles]
[-MAX_SCan_patterns ***pattern_count***]
[-MAX_CHain_patterns ***pattern_count***]
[-TEst_mode_detect {***signal*** **|** ***sequence binary_seq count}***]
[-SHIFT_Control {**Enable** | **Clock** | **None**}]
[-LOAd_unload_cycles ***count1 count2***] [-TAp_controller_interface {**OFf** | **ON**}]
[-Generate_scan_enable {OFf | ON}]

Description

Enables the creation of and specifies the configurations for a low pin count test (LPCT) controller for use with EDT logic.

This command is used during EDT logic creation to create the necessary LPCT controller logic for a test application and during pattern generation to verify the compatibility between the LPCT controller configuration and the design core. This command is also carried forward into the dofile output for ATPG.

Arguments

- **OFF**

  Required literal that disables the creation of an LPCT controller. Default setting.

- **On**

  Required literal that enables the creation of an LPCT controller.

- -MAX_SHift_cycles shift_cycles — Required switch and integer pair that specifies the number of shift cycles in the longest scan chain. This number can be an estimate but it should take into consideration both the EDT and bypass modes. This number affects the size of the shift cycle counter.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- -MAX_CApture_cycles capture_cycles — Required switch and integer pair that specifies the maximum number of capture cycles for all test patterns. This number should take into consideration all of the types of test patterns generated for the test application.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- -MAX_SCan_patterns ***pattern_count***

  Optional switch and integer pair that specifies the maximum number of scan test patterns generated for any test pattern type in the test application. You must consider all the test pattern types used when determining this number.

  This number determines the size of the scan test pattern counter created for the LPCT controller. By default, a counter of size 20 bits is used which implies a maximum of 1048576 scan test patterns.

- -MAX_CHain_patterns ***pattern_count***

  Optional switch that specifies the maximum number of chain test patterns generated for any test pattern type in your test application. You must consider the maximum number of test patterns generated for any chain test type when determining this number. The default size of the chain test pattern counter is 10 bits which implies no more than 1024 chain test patterns.

- -TEst_mode_detect {***signal* | *sequence binary_seq count***}

  Optional switch and literal that configures how the LPCT controller enters test mode. By default, test mode is enabled when the test_mode signal is asserted.

  signal — The Set LPCT Pin test_mode command specifies the name of the signal to be used to enable test mode.

  sequence binary_seq count —The LPCT controller enables test mode if the specified binary sequence, specified by *binary_seq*, is detected within the first *count* cycles after reset.

- -SHIFT_Control {Enable | Clock | None}

  Optional switch and literal pair that specifies how the LPCT controller generates the shift clock control signal. Options include:

  Enable — Alpct_shift_en enable signal is output to generate the shift clock waveform. This is the default setting.

  Clock — A lpct_shift_clock signal is generated when an internal clock is pulsed during shift. Note, if all shift clocks are top-level clocks, they are controllable from top-pins during both shift and capture cycles; in this case, an additional lpct_shift_clock is not needed and is therefore not generated.

  None — No output signal. This option is used when clocks are available at the top-level.

- -LOAd_unload_cycles ***count1 count2***

  Optional switch and double integer that specifies the number of cycles that precede and succeed the shift cycles in the load_unload procedure. Empty cycles corresponding to the specified values are added to the test procedure. This argument must be specified during IP creation.

  These values are determined by the number of cycles before and after the *apply shift* in the the load_unload procedure. Specified values should be higher than the number of cycles in the input test procedure, otherwise they are ignored.

By default, the Type 2 and Type 3 controllers have two pre-shift and two post_shift cycles. By default, the Type 1 controller has zero pre-shift and two post-shift cycles.

- -TAp_controller_interface {**OFf** | **ON**}

  Optional switch and literal that specifies to use a TAP-generated scan enable version of the controller. For more information, see section "Type 2 - LPCT Controller with a TAP."

- -Generate_scan_enable {**OFf** | ON}

  Optional switch and literal that specifies whether the scan_en signal is generated internally by the LPCT controller.

  OFf — If the -TAp_controller_interface argument is set to Off, the LPCT controller has a top-level scan enable signal. This is the default configuration. If the -TAp_controller_interface argument is set to On, an error is issued. For more information, see section "Type 2 - LPCT Controller with a TAP."

  ON — If the -TAp_controller_interface argument is set to Off, the scan enable signal is internally generated by the LPCT controller. If the -TAp_controller_interface argument is set to On, the LPCT the scan enable is generated using the TAP state machine. For more information, see section "Type 2 - LPCT Controller with a TAP."

Example

The following example enables an LPCT controller and configures it to use a sequence to enter test mode:

```
//define parameters for test controller
set_lpct_controller on –max_shift_cycles 400 –max_capture_cycles 2
set_lpct_controller -test_mode_detect sequence 1100100101001110 64
```

Related Commands

report_lpct_configuration
report_lpct_pins
set_lpct_instances

set_lpct_instances

Context: dft -edt

Mode: setup

Usage

set_lpct_instances –Block_location instance_pathname
-Clock_gater_location *instance_pathname*

Description

Inserts the low pin count test (LPCT) controller and clock gater in a specified hierarchical instance within the design core.

The clock gater is added only if set_lpct_controller *-shift_control clock* is specified.

Arguments

- –Block_location instance_pathname

  Optional switch and string pair that specifies an instance in the design core in which to insert the LPCT controller. By default, the LPCT controller is inserted at the top-level of the design core.

- -Clock_gater_location *instance_pathname*

  Optional switch and string pair that specifies an instance in which to insert the LPCT clock gater. By default, the clock gater is inserted at the top-level of the design core.

Example

The following example places the LPCT controller inside the *my_design/xyz/foo* instance in the design core:

```
//define parameters for LPCT controller
set_lpct_controller on –max_shift_cycles 5000 –max_capture_cycles 4 –max_chain_patterns 1000
set_lpct_controller –shift_control clock
set_lpct_instances -block_location my_design/xyz/foo
```

Related Commands

report_lpct_configuration
report_lpct_pins
set_lpct_controller

set_lpct_pins

Context: dft -edt

Mode: setup

Usage

Specify for all configuration

set_lpct_pins
[CLock *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]] |
[CLOCK_Mux_select [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]
[CAPTURE_Enable [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]
[SHIFT_Enable [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]
[SHIFT_Clock [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Specify for Type 1 LPCT controllers (external scan_en)

set_lpct_pins
[INPUT_Scan_enable *pin_name* [*internal_node_name*]
[-ON_controller_module] [-NOInv | -INV]]
[TEST_Clock_connection [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Specify for Type 2 or Type 3 LPCT controllers (internally-generated scan_en TAP or LPCT generated)

set_lpct_pins
[Reset pin_name [*internal_node_name*] [-ON_controller_module]
[-Active High | Low] [-NOInv | -INV]]
[OUTPUT_Scan_enable [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Specify for Type 3 LPCT controller (scan_en generated by LPCT controller only)

set_lpct_pins
[DATA_In *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[TEST_Mode *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[RESET_Out [*pin_name* -ON_controller_module] {*connection_name* | -No_connection}]
[TEST_End [*pin_name* -ON_controller_module] {*connection_name* | -No_connection}]
[TEST_Active [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Specify for Type 2 LPCT controller (scan_en generated using TAP state machine only)

set_lpct_pins
[SHIFT_Dr *pin_name* [*internal_node_name*] [-ON_controller_module]
[-NOInv | -INV]]

[CAPTURE_Dr *pin_name* [*internal_node_name*] [-ON_controller_module]
[-NOInv | -INV]]
[Update_dr *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[-ON_controller_module] [-NOInv | -INV]]
[TMs *pin_name* [*internal_node_name*] [-ON_controller_module] [-NOInv | -INV]]
[TEST_Clock_connection [*pin_name* -ON_controller_module]
{*connection_name* | -No_connection}]

Description

Specifies control pins and connections between the low pin count test (LPCT) controller and the core design.

When using the LPCT controller in which scan_en is generated using TAP states, the clock and reset inputs of the controller should be connected to TCK and the test_logic_reset pins on the TAP controller.

Arguments

- CLock

  Required argument that specifies a free running clock input for a flip-flop based design.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- *pin_name*

  Optional string and switch pair that specify the pin name in the controller RTL for that particular input; this does not specify the connection. Depending on the application, it may be a top-level pin, an input pin, or an output pin as follows. Use this string with the *–on_controller_module* switch to name the output pin when scan cell conditions are used to reset the LPCT controller.

- *internal_node_name*

  Optional string used together with the pin_name argument to specify the top-level pin name and corresponding internal connection for controller input pins. You can use a literal dash "-" for the *pin_name* argument when no top-level pin exists in the design.

- -ON_controller_module

  Optional switch used with the *pin_name* argument that specifies the pin name in the controller RTL for each input; this does not specify the connection.

- *connection_name*

  Optional switch used to specify the names of one or more internal pins to connect to the controller output pins.

- -No_connection

  Optional switch used to specify that no connection is required for a controller output pin.

- -NOInv | -INV

  Optional switch that specifies a LPCT signal is inverted between the point where it is connected during EDT logic creation and the chip boundary in the final netlist used for pattern generation. The default is no inversion.

  You should specify -INV when there is an inversion between the internal connection name for the LPCT pin and the chip boundary such as when specifying an inverting pad.

  Specifying this inversion information during the EDT logic creation phase generates the correct dofiles and test procedure files for the pattern generation phase but has no effect on the EDT logic.

- CLOCK_Mux_select

  Optional argument that specifies the connection to be made to the clock_mux_select output of the controller. This pin is a MUX select signal that selects between the shift clock generated by the LPCT controller and the capture clock. If clock_mux_select is high, the shift clock is selected; if clock_mux_select is low, the capture clock is selected.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- CAPTURE_Enable

  Optional argument that specifies the controller output signal that indicates when the controller is in capture mode. This option specifies the pin to which this capture_enable signal is to be connected.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- SHIFT_Enable

  Optional argument that specifies a controller output signal that indicates when the controller is in shift. The shift_enable option specifies the pins to which the shift_enable signal should be connected. If you provide clock gating using set_lpct_controller *-shift_control enable*, this option is required; it is used as the clock gater enable signal to produce the shift clock from a free running clock.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- SHIFT_Clock

  Argument that specifies the pin is a shift clock signal.

  This option can be used for Type 1, Type 2, and Type 3 LPCT controllers. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- INPUT_Scan_enable

  Argument that specifies the top-level scan enable port.

This is used as an input only to the Type 1 LPCT controller. This option can only be used for the Type 1 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- Reset

  Required argument that specifies the reset input signal for the controller. The reset pin can be shared with the design's functional reset. This can be either an external pin or internal power-on-reset.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- OUTPUT_Scan_enable

  Argument that specifies the list of internal pins to which the LPCT output scan enable pin is connected.

  This option is required when generating a Type 2 or Type 3 LPCT controller.

- DATA_In

  Optional argument that specifies the pin is a data input to the test controller. For the Type 3 controller, a test sequence and configuration is scanned in through this port. This port is usually shared with the edt_channel input.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- TEST_Mode

  Optional argument that specifies the pin is a test mode signal used to enable test mode. This signal is not used when sequence detection is used.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- RESET_Out

  Optional argument that specifies the reset signal to the design. This signal is generated by combining the functional reset signal (specified using "set_lpct_pins reset") with the reset condition or reset control configuration inside the controller. This allows the tool to detect faults connected to the reset signal even while the top-level reset pin is constrained to OFF during capture. If this option is not specified, all the pins driven by the reset signal in the input netlist are changed to be driven by the lpct_reset_out output of the controller. If this option is specified, only the pins explicitly specified are driven by the lpct_reset_out output of the controller.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- TEST_End

  Optional argument that specifies the pin is an end of test signal.

This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- TEST_Active

  Optional argument that specifies an output signal from that LPCT controller that indicates the LPCT controller is active and controlling the scan test.

  This option is only valid for the Type 3 controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- SHIFT_Dr

  Required argument that specifies the pin is a Shift_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- CAPTURE_Dr

  Required argument that specifies the pin is a Capture_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- Update_dr

  Required argument that specifies the pin is an Update_Dr signal on the TAP controller.

  This option can only be used for the Type 2 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- TMs

  Required argument that specifies the pin is a test mode select tms input pin of the TAP.

  This option can only be used for the Type 2 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- -Active {**High**|**Low**}

  Optional switch and literal pair that specifies the active state for the input reset signal. By default, the reset is active high.

  This option can only be used for the Type 3 LPCT controller. For more information, see "LPCT Controller Types" in the *Tessent TestKompress User's Manual*.

- TEST_Clock_connection

  Optional switch that specifies the test clock input pin for the user-added test_mode mux. The tool connects the output of the clock gater, added to facilitate sharing of the LPCT clock and the scan clock, to the specified pin. The user must connect the other input of the mux to the functional clock and the the select pin to the test mode signal.

Example 1

The following example defines the connections between the LPCT controller and the design core.

```
// Temporary top-level scan IO pins (removed by TK during IP creation
add_scan_chains chain1 grp1 scan_in1 scan_out1
add_scan_chains chain2 grp1 scan_in2 scan_out2

// LPCT controller input pins
set_lpct_pins clock ref_clk clk_pad/Z
set_lpct_pins reset_out - pwr_up/rst
set_lpct_pins data_in fnl_input

// Connections from LPCT logic outputs to design
set_lpct_pins output_scan_enable scan_en
set_lpct_controller -shift_control enable
set_lpct_pins shift_en clk_mux/shift_en
set_lpct_pins capture_en clk_mux/cap_en
```

Example 2

The following example defines the connections between the LPCT controller and the design core when the LPCT controller internally generates the scan_en signal.

```
set_lpct_controller -generate_scan_enable on -max_shift 1000 -max_capture 3
set_lpct_pins output_scan_enable scan_mode
```

Example 3

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists.

```
set_lpct_controller on
set_lpct_pins input_scan_enable scan_en
```

Example 4

The following example defines the connections between the LPCT controller and the design core when a top-level scan_en signal exists and the user has implemented clock control.

```
set_lpct_controller -shift_control enable
set_lpct_pins capture_en pll/capture_start
set_lpct_pins clock_mux_select pll/select
set_lpct_pins shift_en pll/shift_en
```

Example 5

The following example defines the connections between the LPCT controller and the design core when a TAP generated scan_en signal exists.

```
set_lpct_controller -tap_controller on -generate_scan_enable on
set_lpct_pins output_scan_enable scan_mode
set_lpct_pins shift_dr - tap/shiftEn
set_lpct_pins capture_dr - tap/captureEn
set_lpct_pins update_dr - tap/updateEn
set_lpct_pins test_mode - tap/atpgInstruction
set_lpct_pins reset - tap/resetl -active low
```

Example 6

The following example defines the connections between the LPCT controller and design core when a TAP is used to generate scan_en signal

```
set_lpct_pins clock refclk -on_controller_module          // input pin
set_lpct_pins capture_en pll_start -on_controller_module // output pin
```

Example 7

The following example specifies to use a top-level scan clock as the LPCT clock for a Type 1 controller.

```
set_context dft –edt
add_clock 0 clk                    // There is a single clock in the design
add_scan_chains ...
set_lpct_controller on –shift_control clock
set_lpct_pin clock clk             // LPCT clock is shared with scan clock
set_lpct_pin input_scan_enable scan_en
set_lpct_pin test_clock_connection test_mode_mux/B
set_system_mode analysis
report_lpct_pins
write_edt_files created -replace
```

Example 8

The following example specifies to use tck as the scan shift clock in a Type 2 LPCT controller. Note, the test mode mux that selects between the LPCT-generated scan clock and the functional clock already exists in the design. The test clock connection pin on the mux is specified with the test_clock_connection pin type.

```
set_context dft –edt
add_clock 0 tck
add_scan_chains ...
set_lpct_controller –tap_controller_interface on
set_lpct_controller –shift_control clock
set_lpct_pins output_scan_enable scan_en
set_lpct_pins clock tck pad_tck/Z      // LPCT clock is tck
set_lpct_pins tms tms pad_tms/Z
set_lpct_pins reset – tap_i/tlr
set_lpct_pins capture_dr – tap_i/capturedr
set_lpct_pins shift_dr – tap_i/shiftdr
set_lpct_pins update_dr – tap_i/updatedr
set_lpct_pins test_mode – tap_i/edt_scan_inst
set_lpct_pins test_clock_connection test_mode_mux/B
set_edt_options –channel 1
set_edt_pins input 1 tdi pad_tdi/Z
set_edt_pins output 1 tdo tap_i/tap_edt_channel_reg_in
set_system_mode analysis
report_lpct_pins
report_lpct_configuration
write_edt_files created -replace
```

Related Commands

report_lpct_configuration
report_lpct_pins
set_lpct_controller

set_macrotest_options

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

Mode: setup, analysis

Usage

set_macrotest_options **{OFf | ON}**

Description

Prepares a black box macro to allow MacroTest conversion of patterns at the boundaries of that instance.

The set_macrotest_options command changes two normal behaviors of the tool's DRC. To obtain a pulse from a macro, a PI of the circuit that has been declared as a clock, read control, or write control must drive the pin of the macro. If the black box is an empty instantiation of a RAM, it is likely that the tool needs to pulse the write control to write into the RAM. Normally, if a write control does not connect to a _ram or _cram primitive, DRC issues a G5 violation and does not take you out of Setup until the write control is deleted. The set_macrotest_options command prevents this and treats any write control as valid (as if it were connected to a _ram or _cram).

Another DRC aspect that the tool modifies occurs if a transparent latch (TLA) is on the control (input) side of a black box macro, and is unobservable due to the macro. Tessent FastScan converts a black box macro to a set of TieX gates, one per output. As a result, if the TLA only fans out to the macro, the tool cannot observe it (it has no sensitizable path to some PO or scan cell). Normally, DRC converts such a TLA to a TieX gate, preventing MacroTest from justifying values on the input side of the black box macro backward through the TLA. This is because it thinks the TieX means the primitive cannot be controlled. The set_macrotest_options command prevents this and keeps unobservable TLAs as TLAs.

If in doubt, issue the set_macrotest_options command in setup mode before proceeding with macro testing. No errors occur if DRC does not encounter the situations resolved by this command. Therefore, you can issue the set_macrotest_options command before every MacroTest run if you desire.

Arguments

- **{OFf | ON}**

  A required Boolean that specifies whether to modify DRC as described in the preceding description.

  OFf — Disables DRC modification (normal DRC is performed). This is the default at invocation.

  ON — Enables modified DRC analysis to avoid potential macrotest-specific ATPG issues.

Example

The following example shows the command issued to allow black box macro testing:

set_macrotest_options on

Related Commands

macrotest

set_misr_connections

Context: patterns -scan

Mode: setup

Usage

set_misr_connections ***misr_name*** **-tap** ***integer*** **-chain** ***chain_name***…

Description

Describes the connections between the scan chain outputs and the MISR.

This command is used with the hybrid TK/LBIST flow—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

If you map multiple scan chains to the same MISR position, then the tool infers a XOR spatial compactor that maps from those multiple scan chain outputs to the specified MISR feedback position.

Arguments

- ***misr_name***

  A required string that specifies the name of the MISR specified with the add_misrs command.

- **-tap** ***integer***

  A required switch and required and repeatable integer that specifies the tap points for the MISR feedback network. The tap index starts from 1, to be consistent with the decompressor commands.

- **-chain** ***chain_name***

  A required switch and string that specifies the name of the scan chain pin that you want to connect to the MISR specified by ***misr_name***.

Example

This example describes the hardware connection from scan chain outputs to MISR inputs through a 4-to-1 XOR compactor.

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[0] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[1] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[2] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[3] -tap 1
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[4] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[5] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[6] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[7] -tap 2
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[8] -tap 3
```

```
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[9] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[10] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[11] -tap 3
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[12] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[13] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[14] -tap 4
set_misr_connection misr -chain /my_core_edt_i/my_core_edt_compactor_i/edt_scan_out[15] -tap 4
```

Related Commands

add_misrs

delete_misrs

set_module_matching_options

Context: all contexts

Mode: all modes

Usage

set_module_matching_options [-prefix_pattern_list *prefix_pattern_list*]
[-suffix_pattern_list *suffix_pattern_list*] [-regexp] [-nocase] [-append]

Description

Defines the acceptable prefixes and suffixes or regular expressions to use when matching an ICL module to a design module during ICL extraction.

During ICL extraction, the list of all design modules found below the current design is used to search for corresponding ICL modules. The ICL modules are then attached to the design modules and the connectivity between them is traced and written in a ICL module description.

The assumption is that for every instrument you want to trace in the design module, you must have an ICL description with the same module name. The ports found in the ICL module must be found in the design module otherwise I1 DRC violations will be issued.

If the instrument was inserted in RTL and ICL extraction was run on the synthesized view, the instruments that are used more than once are likely to have been uniquified with an integer suffix. You may also have instructed the synthesis tool to prefix every synthesized module with a common prefix.

Arguments

- -prefix_pattern_list *prefix_pattern_list*

  An optional switch and string pair that specifies the prefix patterns taken into account when matching ICL modules to design modules. The *prefix_pattern_list* string or Tcl list of strings lists the prefix strings or regular expressions to be used as a prefix to the ICL module name when matching it against the design module names.

- -suffix_pattern_list *suffix_pattern_list*

  An optional switch and string pair that specifies the suffix strings or regular expressions to be used for uniquified design module mapping. This switch can only be used once. The *suffix_pattern_list* string or Tcl list of strings lists one or more patterns to be used for uniquified design module mapping. By default, the tool matches any module name with a suffix that matches the regular expression “{_[0-9]+}”. This means that, by default, the tool will match all names of the format <module_name>_<number>.

- -regexp

  An optional switch that directs the matching patterns as a regular expression instead of as a simple wildcard pattern. See section “Glob and Regular Expression Pattern Matching Syntax” on page 107 for a description of the regular expression syntax.

- -nocase

  An optional switch that specifies that the pattern matching is case insensitive.

- -append

  An optional switch that specifies to append the newly specified pattern list to the existing list or to replace the existing list.

Return Value

None

Example

In the following example, this command is used to allow matching ICL modules with an optional prefix "mycore_" and an optional suffix matching _ts[0-9]+. In this example, an ICL module IntrumentA will be allowed to match mycore_InstrumentA_ts1 and mycore_InstrumentA_ts2.

```
set_module_matching_options -prefix_pattern {mycore_} -suffix_patterns {_ts[0-9]+} -regexp
```

Related Commands

report_module_matching

report_module_matching_options

set_multiple_detection

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

Mode: setup, analysis

Usage

set_multiple_detection
[-Guaranteed_atpg_detections *guaranteed_atpg_detections*]
[-Desired_atpg_detections *desired_atpg_detections*]
[-Simulation_drop_limit *simulation_drop_limit*]
[-REPort_incremental_statistics {ON | OFF}] [OFF]

UDFM Usage:

set_multiple_detection
[-Simulation_drop_limit *simulation_drop_limit*] [OFF]

Description

Enables the multiple detection of faults.

Multiple detection is a statistical way to gain bridge fault coverage. By default, the tool generates a single detection pattern set, where a fault is detected once, then dropped from the simulation.

When multiple detection is enabled, the DS fault category represents faults detected one or more times, not just faults detected the threshold number of times. The create_patterns command includes a cumulative Bridge Coverage Estimate (BCE) in the statistics transcription for each batch of patterns. Use the *simulation_drop_limit* argument to increase the accuracy of the BCE, being aware that the improved accuracy entails more run time.

The BCE, which is based on simulation results, is calculated as follows:

$$\mathrm{BCE} = \left[\sum_{i=1}^{n} \frac{f_i}{TE} \cdot (1 - 2^{-i}) \right] + \frac{N_{DI}}{TE}$$

where:

f_i = number of faults detected *i* times by the test pattern set
TE = number of total testable faults in the target fault list
N_{DI} = number of DI faults in the target fault list (includes protected faults)
n = *atpg_detections* (or *fault_drop_detections* if used)

Enabling multiple detections changes the behavior of the following commands:

- analyze_compression — If the -Preserve_faults_from_analysis switch is specified, preserves the N-detection information for all the detected faults during the analyze_compression operation.

- create_patterns — Performs multiple iterations (loops) as needed to ensure every testable fault is detected *guaranteed_atpg_detections* times and includes a cumulative bridge coverage estimate (BCE) in the statistics transcription for each batch of patterns simulated. Each pattern counts at most only one detection for any particular fault.
- report_statistics — Lists the final BCE in reports and additionally includes a histogram that shows the profile of the number of detections for multiple detection DS faults.
- report_environment — Lists a "multiple detection = …" line that shows the current multiple detection settings. If this line is absent, it means multiple detection is disabled.
- report_faults — Displays a "num_detections..." line for each DS fault that shows the number of detections for that fault, except when fault protection is in effect.
- write_faults — Writes a "num_detections..." line for each multiple detection DS fault that shows the number of detections for that fault, except when fault protection is in effect.

Note

For information about fault protection, refer to the descriptions of these commands: "read_faults -Protected" and set_fault_protection.

Issue this command without any arguments to report the current setting. If you issue this command with arguments more than once, the last command overwrites the previous settings.

Note that when the fault type is UDFM, the set_multiple_detection command works only for simulation and not for ATPG.

Arguments

- -Guaranteed_atpg_detections *guaranteed_atpg_detections*

  An optional switch and integer pair that specifies the number of detections required for each testable fault. The *guaranteed_atpg_detections* can be any integer between 1 and 255 and determines the number of detections for each fault that the create_patterns command should target during pattern generation. The specified number of detections for each fault are guaranteed during fault simulation before the tool drops the fault from the fault list.

  The default value of *guaranteed_atpg_detections* is 1. Increasing the number of guaranteed detections can significantly increase the pattern count and ATPG run time.

- -Desired_atpg_detections *desired_atpg_detections*

  An optional switch and integer pair that specifies the number of detections desired for each testable fault. These faults are detected using Embedded Multi-Detect (EMD) functionality. The *desired_atpg_detections* can be any integer between 1 and 255.

  Setting *desired_atpg_detections* to 1 disables EMD.

  By default, *desired_atpg_detections* is equal to *guaranteed_atpg_detections*.

The *desired_atpg_detections* cannot be less than the *guaranteed_atpg_detections* except when EMD is disabled (*desired_atpg_detections* set to 1).

Unlike the number of guaranteed detections, increasing the number of desired detections does not increase the pattern count. However, it may increase the ATPG run time. For optimal performance, it is not recommended to set the *desired_atpg_detections* to greater than 5.

- -Simulation_drop_limit *simulation_drop_limit*

  An optional switch and integer pair that specifies the maximum number of detections for a fault. The tool drops the fault from the simulation after the *simulation_drop_limit*.

  When the fault type is UDFM, you can set the *simulation_drop_limit* to any positive integer. By default, the *simulation_drop_limit* is 1.

  For all other fault types, the *simulation_drop_limit* cannot be less than the *guaranteed_atpg_detections* or *desired_atpg_detections* values. If you do not specify a value, the *simulation_drop_limit* is automatically set to one of the following:

  - ***guaranteed_atpg_detections* = *desired_atpg_detections* = 1**

    The *simulation_drop_limit* is set to 1.

  - ***guaranteed_atpg_detections* <= 10** AND **2** <= ***desired_atpg_detections* <= 10**

    The *simulation_drop_limit* is set to 10.

  - ***guaranteed_atpg_detections* > 10** OR ***desired_atpg_detections* > 10**

    The *simulation_drop_limit* is set to the greater of the *guaranteed_atpg_detections* or *desired_atpg_detections* values.

- -REPort_incremental_statistics {ON | OFF}

  An optional switch and literal pair that enables the tool to report incremental multiple-detect test coverage statistics at the end of each multiple-detect ATPG loop. The default is for the tool to not report such statistics.

- OFF

  An optional literal that disables the multiple detection of faults.

Example 1

The following example generates a 3-detection pattern set without EMD, where a fault is dropped from the simulation after 5 detections:

```
set_multiple_detection -guaranteed_atpg_detections 3 -desired_atpg_detections 1
    -simulation_drop_limit 5
```

Example 2

The following example generates a pattern set where each testable fault must be detected twice and should be detected at least 8 times:

```
set_multiple_detection -guaranteed_atpg_detections 2 -desired_atpg_detections 8
```

The simulation drop limit is automatically set to 10.

Example 3

The following example disables the multiple detection of faults:

```
set_multiple_detection off
```

Related Commands

create_patterns
read_faults
report_environment
report_faults
report_statistics
set_fault_protection
set_fault_type
write_faults

set_multiprocessing_options

Tessent Shell Supported Contexts and Modes:

Context: unspecified, dft -edt, patterns -scan

Mode: setup, analysis, insertion

Usage

set_multiprocessing_options [-generic_delete *string*] [-generic_scheduler *string*]
[-license_timeout *integer*] [-lsf_heuristics { on | off }] [-lsf_learning { on | off }]
[-lsf_options *string*] [-multithreading { on | off }]
[-processors_per_grid_request *number*] [-remote_shell {rsh | ssh}]
[-result_time_limit *number*] [-scheduler_timeout *number*] [-sge_options *string*]

Description

Sets the values of several variables that affect multiprocessing commands for ATPG or simulation.

Note

Take care to exclude any unsupported platforms that exist in your grid environment by specifying appropriate grid job submission constraints. For a list of supported platforms, refer to "Supported Hardware and Operating Systems" in *Managing Mentor Graphics Tessent Software*.

For more information about setting up for multiprocessing, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual* and "Using the Tessent Tcl Interface" in the *Tessent Shell User's Manual*.

Arguments

- -generic_delete *string*

  An optional switch and string that specifies a command string to delete a job. The tool uses this command to delete jobs when they are no longer needed, including cases when the scheduler exceeds its timeout setting or is interrupted by Ctrl-C.

- -generic_scheduler *string*

  An optional switch and string that specifies a command script that requests a slave process from the generic job scheduler (that is used for grid systems other than LSF or SGE). The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent add_processors commands, not currently running processes. An incorrect specification can prevent the correct operation of the add_processors command.

- -license_timeout *integer*

  An optional switch and integer that specifies the number of minutes for the tool to wait when attempting to acquire a license. The default value is 5 minutes. If you specify "0," there is no time limit and the tool waits until a license is available.

- -lsf_heuristics { on | off }

  An optional switch and keyword that enables or disables LSF heuristics learning when you add a processor by LSF grid request. The tool uses LSF heuristics learning in an attempt to automatically choose machines on the LSF grid that are compatible with the tool if you did not specify an LSF "type" constraint with -lsf_options and if either LSF scheduler learning has failed or you disabled that feature using "-lsf_learning off."

  LSF heuristics learning may be able to determine tool compatibility based on the names of the types and models available and successfully identify some type/model combinations that the tool can use; however, this may also result in specifying incompatible combinations or miss some compatible combinations. If LSF heuristics learning is turned off or fails, then the tool submits the job anyway without specifying any type/model restrictions.

  LSF heuristics learning is enabled by default. For more information about using this switch, refer to "Adding Processors to the LSF Grid" in the in the *Tessent Scan and ATPG User's Manual*.

- -lsf_learning { on | off }

  An optional switch and keyword that enables or disables LSF scheduler learning when you add a processor by LSF grid request. When this switch is on and you have not specified an LSF "type" constraint with -lsf_options, the tool asks the LSF system what machines and type/model combinations are available, after which the tool runs the **lsrun** command to log onto likely machines to determine which are compatible with the tool.

  LSF scheduler learning cannot succeed if your LSF system is configured to prevent use of **lsrun**. If LSF scheduler learning is turned off or fails, the tool then uses LSF heuristics learning unless you have disabled this feature with "-lsf_heuristics off."

  LSF scheduler learning is enabled by default. For more information about using this switch, refer to "Adding Processors to the LSF Grid" in the in the *Tessent Scan and ATPG User's Manual*.

- -lsf_options *string*

  An optional switch and string that specifies command options for the tool to append to the job submission command for the LSF job scheduler. The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent add_processors commands, not to currently running processes. An incorrect specification can prevent the correct operation of the add_processors command.

  If you add the "verbatim" key symbol at the beginning of the string, the tool interprets the rest of the line as the submission specification and performs no evaluation. For more information, refer to "Example 8" on page 1845.

- -multithreading { on | off }

  An optional switch and keyword that disables multithreading functionality. Multithreading is enabled by default.

- -sge_options *string*

  An optional switch and string that specifies command options for the tool to append to the job submission command for the SGE or OGE job scheduler. The string value you set for this switch replaces the current value. Setting a correct value for the string requires a knowledge of the scheduler configuration at your site and the job control submission syntax. Applies only to subsequent add_processors commands, not currently running processes. An incorrect specification can prevent the correct operation of the add_processors command.

- -scheduler_timeout *number*

  An optional switch and number value that specifies a time limit in minutes for attempts to add_processors using a job scheduler. If the tool does not receive all the requested processors within the specified time, it will proceed with the processors it has received. The default number is 10 minutes.

- -remote_shell {rsh | ssh}

  An optional switch and keyword that specifies the shell command that the master host should use to create processes on slave hosts. Applies only when you specify hosts manually rather than using a job scheduler. By default, the tool uses the **rsh** command.

- -result_time_limit *number*

  An optional switch that specifies a time limit in minutes for detecting non-responsive slaves while waiting for certain types of job results. If the tool detects that the time limit has been exceeded for a specific result needed to make further progress, the tool issues a warning about the non-responsive slave and waits again for the same result. (This gives you the opportunity to investigate and correct the problem.) If the result_time_limit is again exceeded, the tool kills the slave and reassigns the job to another processor, and the cycle starts over. In most cases, you do not have to change the default value. Note that if you set the value too low, the tool could possibly kill all of its slaves needlessly. A value of 0 disables the time limit, but otherwise the command does not accept a value of less than 15. The default value is 45.

- -processors_per_grid_request *number*

  An optional switch and number value that specifies the maximum number of processors (slots) to group into one single request for the grid system, either for SGE, LSF, or GENERIC requests. The default value is “unspecified” (-1) because there are different default values for SGE, LSF, and GENERIC. The default for LSF requests is 4. The tool can use LSF job submission switches to ensure that this number of processors is requested per grid request. For SGE and GENERIC processor requests, the syntax for requesting a specific number of processor slots per grid request is site-specific, so the tool defaults to one processor per request.

If you want to specify the number of processors per request when using SGE or a generic grid scheduler, you must supply the needed syntax using the -sge_options, -generic_scheduler, and (optionally) the -generic_delete switches. You can do this by hardcoding a value in the -sge_options or -generic_scheduler string, or by using the pseudo variable *%processors_per_grid_request*, which allows you to change the value of processors_per_grid_request later without also having to change the corresponding grid command variable value. For an example of using this variable for the -lsf_options switch, refer to "Example 3" on page 1844.

The requested number of processors (for example, add_processors sge:20) results in the following number of grid requests: the requested number of processors divided by processors_per_grid_request. The result of the division is rounded up, and the last grid request can request fewer processors than the number of "processors_per_grid_request."

Note

The use of job schedulers requires certain environment prerequisites to be satisfied before you invoke the tool. For more information, refer to "Using Multiprocessing to Reduce Runtime" in the *Tessent Scan and ATPG User's Manual*.

Command Variables

The following variables are available for use in this command. You can define these variables in your dofile for convenience.

- %command — Represents a substring you use with the -generic_scheduler switch to specify the location to substitute the Mentor Graphics command script that launches a slave process. For more information, refer to "Example 5" on page 1844.
- %job — Specifies where to insert the actual job number in the **qdel** command syntax and also defines the string "job" to search for in the generic_scheduler's output. For more information, refer to "Example 6" on page 1845.
- %processors_per_grid_request — Specifies the maximum number of processors (slots) to group into one single request for the grid system, either for SGE, LSF, or GENERIC requests. For more information, refer to "Example 3" on page 1844.

Tcl Notes

When using Tcl constructs with the set_multiprocessing_options command, the tool's logfile reports the evaluated command and strips any Tcl special characters (quotes, braces, and brackets) from the logfile. If you need to replay a session from the logfile, you must add the Tcl syntax.

Enclosing any expression in braces ({ }) prevents the tool from evaluating the expression. Enclosing any expression in double quotes (" ") causes the tool to resolve any variables in the expression.

Example 1

The following example controls type and memory constraints that would normally be supplied automatically:

```
set_multiprocessing_options -lsf_options { -R \
    select[type==LINUX64 && mem> 200 && swp > 200 ] }
get_multiprocessing_option -lsf_options
-R select[type==LINUX64 && mem> 200 && swp > 200]
```

This setting would constrain LSF to select hosts meeting the specified requirements. The “type” specification also inhibits the automatic learning that would normally take place to discover all the type/model names in an LSF cluster. (This is useful if your site prevents the use of the **lsrun** command or if the learning hits non-responsive hosts.) You can specify any legal LSF constraints for the installation, which overrides any related specifications the tool would have provided otherwise. It is up to you to supply syntactically and semantically correct LSF options.

Example 2

The following example specifies that LSF use queue *myqueue*:

```
set_multiprocessing_options -lsf_options "-q myqueue"
```

Example 3

The following example specifies that LSF reserve *%processors_per_grid_request* processors or slots:

```
set_multiprocessing_options -lsf_options "-n  %processors_per_grid_request"
```

Note, however, that the -n specification would probably be only a subset of the bsub options assigned to the -lsf_options variable, and that the above example command line serves only as an example of how to use the *%processors_per_grid_request* variable. If you don't use the -n switch in the -lsf_options string, the tool still specifies -n with the processors_per_grid_request value to the LSF job scheduler by default (value of 4). Note than when requesting additional processors per grid request, the number of grid requests decreases accordingly.

Example 4

The following example specifies that SGE impose a 6 hour time limit on slave processes:

```
set_multiprocessing_options -sge_options {-l h_rt=6:00:00}
```

Example 5

The following example specifies an alternative interface to SGE for launching slave processes:

```
set_multiprocessing_options -generic_scheduler \
    { $SGE_ROOT/bin/lx*-amd64/qsub %command }
```

The *%command* represents a substring you must use with the -generic_scheduler switch to specify the location to substitute the Mentor Graphics command script that launches a slave process. Variables can be defined in the dofile to make this more convenient, as in the following example dofile excerpt:

```
set SUBMIT { $SGE_ROOT/bin/lx*-amd64/qsub }
set_multiprocessing_options -generic_scheduler "${SUBMIT} %command"
```

Example 6

The following example specifies an alternative interface to SGE for terminating slave processes:

add_processors generic

set_multiprocessing_options -generic_delete { $SGE_ROOT/bin/lx24-x86/qdel *%job* }

The *%job* defines where to insert the actual job number in the **qdel** command syntax and also defines the string "job" to search for in the generic_scheduler's output.

Example 7

The following example specifies that the master host should use the **ssh** shell command to create processes on slave hosts (manual specification of hosts only):

set_multiprocessing_options -remote_shell ssh

Tip: If you encounter errors trying to use SSH, refer to "Troubleshooting SSH Environment and Passphrase Errors" in the *Tessent Scan and ATPG User's Manual*.

Example 8

The following example specifies that LSF job requests should specify the use of hosts with type LINUX64 and should not specify any additional job constraints:

set_multiprocessing_options -lsf_options verbatim { -R "select[type==LINUX64]" }

Related Commands

add_processors
delete_processors
get_multiprocessing_option
report_multiprocessing_options
report_processors

set_name_prefix

Context: dft -scan

Mode: setup, analysis

Usage

set_name_prefix [{-Net *prefix_name*} |
{-INStance{[Tristate|Xbound|Lockup|IN_register|OUt_register]*prefix_name*}…}
{-Scan_chain *prefix_name*} |
{-INPut_wrapper_chain *prefix_name*} |
{-Output_wrapper_chain *prefix_name*}]

Description

Explicitly defines the default names for nets and instances, and reports current or modified settings.

The set_name_prefix command serves two purposes. You can use it to change the default prefix that the tool uses to name certain types of added test logic, and you can change the default the tool uses to name nets.

If you invoke the command without an argument, it automatically reports on the current settings for all prefixes. If you make any changes, it reports the modified settings. Table 7-43 shows the invocation defaults for particular logic instance types:

Table 7-43. Instance Type Prefix Defaults

Object Type	Default Prefix Name	Instance Literal
Tri-state control	tcntl	Tristate
X bounding	xbnd	Xbound
lockup cells	lckup	Lockup
Input partition flip-flop	inreg	IN_register
Output partition flip-flop	outreg	OUt_register
Other logic	uu	

Arguments

- -Net *prefix_name*

  An optional switch and string pair that specifies the prefix_name you want as the default prefix for naming nets. The invocation default prefix is net.

- -INStance [Tristate | Xbound | Lockup | IN_register | OUt_register] *prefix_name*

  An optional switch, with an optional, repeatable literal and string pair, that specifies the prefix_name you want as the default prefix for the object type specified by the literal. If you do not specify an object type literal, prefix_name applies to all added test logic not covered by one of the object type literals. The default prefix for this other logic is uu.

Tristate — An optional literal that specifies to apply the prefix_name as the default for the logic object type tri-state control. The default prefix is tcntl.

Xbound — An optional literal that specifies to apply the prefix_name as the default for the logic object type X bound control. The default prefix is xbnd.

Lockup — An optional literal that specifies to apply the prefix_name as the default for the logic object type lockup cell. The default prefix is lckup.

IN_register — An optional literal that specifies to apply the prefix_name as the default for the logic object type input partition flip-flop. The default prefix is inreg.

OUt_register — An optional literal that specifies to apply the prefix_name as the default for the logic object type output partition flip-flop. The default prefix is outreg.

- -Scan_chain *prefix_name*

  An optional switch and string pair that specifies a new default prefix for naming scan chains of any scan chain type. At invocation, the default value for prefix_name is chain.

- -INPut_wrapper_chain *prefix_name*

  An optional switch and string pair that specifies a new default prefix for naming input wrapper chains. At invocation, the default value for prefix_name is chain.

- -Output_wrapper_chain *prefix_name*

  An optional switch and string pair that specifies a new default prefix for naming output wrapper chains. At invocation, the default value for prefix_name is chain.

Example 1

The following example resets the default prefix name for tri-state control logic to “tric” and for lockup cells to “lockl”:

set_name_prefix -instance tristate tric lockup lockl

The tool will change the prefix names and issue the following report:

```
//  set_name_prefix prefixes:
//    nets             : net
//    tristates        : tric
//    xbounding        : xbnd
//    lockup cells     : lockl
//    input registers  : inreg
//    output registers : outreg
//    other instances  : uu
```

Example 2

The following example overrides the default scan chain name assignments for all three scan chain types and applies the new default prefixes to the newly created chains of each type.

set_name_prefix -scan_chain coreChain
set_name_prefix -input_wrapper_chain inputWrapper
set_name_prefix -output_wrapper_chain outputWrapper
...
report_scan_chains

```
Input wrapper chains:
---------------------------------------------------------------------------
chain = inputWrapper1 group = group1 input = /scan_in1 output = /out1
  length = 3 scan_enable = /se_in  clock = /clk
chain = inputWrapper2 group = group1 input = /scan_in2 output = /scan_out1
  length = 3 scan_enable = /se_in  clock = /clk
chain = inputWrapper3 group = group1 input = /scan_in3 output = /scan_out2
  length = 2 scan_enable = /se_in  clock = /clk

Output wrapper chains:
---------------------------------------------------------------------------
chain = outputWrapper1 group = group1  input = /scan_in4  output = /out2
  length = 1 scan_enable = /se_out  clock = /test_clk

Default scan partition chains:
---------------------------------------------------------------------------
chain = coreChain1 group = group1  input = /scan_in5  output = /scan_out3
  length = 1 scan_enable = /scan_en  clock = /clk
```

Related Commands

insert_test_logic
set_test_logic
report_scan_chains
set_registered_io

set_net_dominance

Context: all contexts

Mode: setup, analysis (dft -edt and patterns -scan contexts only)

Usage

set_net_dominance **Wire** | **And** | **Or**

Description

Specifies the fault effect of bus contention on tri-state nets.

The set_net_dominance command specifies the fault effect of bus contention on tri-state nets. This provides the capability to detect some faults on tri-state driver enable lines when those drivers connect to a tri-state bus. These faults would be ATPG-untestable unless the tool can use the Z-state for detection.

When using Tessent FastScan or Tessent TestKompress, the Wire behavior is the same where any different binary value results in an X-state.

The truth tables for each type of bus contention fault effect are given in Tables 7-44, 7-45, and 7-46. This command does not affect Good machine behavior.

Table 7-44. WIRE Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	X	X	X
0	X	0	X	0
1	X	X	1	1
Z	X	0	1	Z

Table 7-45. AND Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	0	X	X
0	0	0	0	0
1	X	0	1	1
Z	X	0	1	Z

Table 7-46. OR Bus Contention Truth Table

	X	**0**	**1**	**Z**
X	X	X	1	X
0	X	0	1	0
1	1	1	1	1

Table 7-46. OR Bus Contention Truth Table

	X	**0**	**1**	**Z**
Z	X	0	1	Z

Arguments

- **Wire**

  A literal that specifies for the tool to use unknown behavior for the fault effect of bus contention on tri-state nets. This is the default behavior upon invocation of the tool.

- **And**

  A literal that specifies for the tool to use wired-AND behavior for the fault effect of bus contention on tri-state nets.

- **Or**

  A literal that specifies for the tool to use wired-OR behavior for the fault effect of bus contention on tri-state nets.

Example

The following Tessent FastScan example specifies that the fault effect on tri-state nets is wired-AND during ATPG:

```
set_net_dominance and
set_system_mode analysis
create_patterns
```

set_net_resolution

Context: all contexts

Mode: setup

Usage

set_net_resolution **Wire** | **And** | **Or**

Description

Specifies the behavior of multi-driver nets.

The set_net_resolution command specifies the behavior of non-tri-state, multi-driver nets. The default upon invocation of the tool is Wire, which requires all inputs be at the same value to achieve a value. If you can model your nets using the And or Or option, you can improve your test coverage results.

Arguments

- **Wire**

  A literal that specifies for the tool to use unknown behavior for non-tri-state, multi-driver nets. This requires all inputs to be at the same value to achieve a value other than X. This is the default upon invocation of the tool.

  If you are using Tessent FastScan or Tessent TestKompress with E9 rule checking enabled, the tools do not perform checking on wire gates if you use the set_net_resolution command to change their behavior to AND or OR.

- **And**

  A literal that specifies for the tool to use wired-AND behavior.

- **Or**

  A literal that specifies for the tool to use wired-OR behavior.

Example

The following Tessent FastScan example specifies that the behavior of non-tri-state, multi-driver nets is wired-AND during ATPG:

```
set_net_resolution and
set_system_mode analysis
create_patterns
```

Context: dft -scan

Mode: setup

set_number_shifts

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

Mode: setup, analysis

Usage

set_number_shifts ***shift_number***

Description

Sets the number of shifts for loading or unloading the scan chains.

Note

For diagnosis, you can only use the set_number_shifts command to modify the number of shifts for an EDT design. There is currently no support in the diagnosis tool for this for non-EDT designs.

The number of shifts used for loading or unloading the scan chains in a scan group is the same as the largest number of scan cells in any scan chain in that scan group. The tool determines this number once it traces all scan chains. This is the default. You can use this command to increase the default number.

This command is normally issued in Setup mode. If the specified number of cycles is more than the number determined during DRC, the additional cycles are added as overshift cycles.

If EDT is enabled, you can issue this command in either setup or analysis mode. The number of shift cycles specified in analysis mode does not impact the pattern generation process which uses the number determined during DRC. The generated patterns are padded with the additional cycles when they are saved using the write_patterns command.

The set_number_shifts command is mutually exclusive with the set_edt_options -*OVERShift_cycles integer* switch; when either command is issued, a check is performed to verify that the other command or command and switch has not been issued. If they are used together, an error message is issued and the number of shifts is reset to zero.

If the user-specified shift cycle is less than required, an error is issued.

Note

The set_number_shifts command has no effect on WGL patterns if both the WGL_FULL_CHAIN and WGL_FULL_SCANGROUP parameter file keywords are set to 0. By default, WGL_FULL_CHAIN is set to 1 and WGL_FULL_SCANGROUP is set to 0.

Arguments

- ***shift_number***

  Specifies the number of shifts. If the number specified is smaller than the default number determined by the tool, the tool issues an error message.

Example 1

The following example sets the number of shifts for loading or unloading the scan chains to 2.

```
set_number_shifts 2
```

Example 2

The following example adjusts the number of shifts for a given pattern set and runs diagnosis:

```
set_number_shifts 100
read_patterns my_patterns/pat.wgl
diagnose_failures flogs/fail_1.log
```

Related Commands

set_edt_options

set_observation_point

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

Mode: setup, analysis

Usage

set_observation_point **Master** | **SLave** | **SHadow** | **Clockpo**

Description

Specifies the observation point for random pattern fault simulation.

The set_observation_point command specifies whether Tessent FastScan observes master latches, slave latches, shadow latches, or clock primary outputs during random pattern fault simulation. If you select Master, Slave, or Shadow, Tessent FastScan also observes the primary outputs that do not connect to clock lines. The default behavior upon invocation of Tessent FastScan is Master.

Arguments

- **Master**

  A literal that specifies observation of master latches and normal primary outputs. This is the default behavior upon invocation of Tessent FastScan.

- **SLave**

  A literal that specifies observation of slave latches and normal primary outputs.

- **SHadow**

  A literal that specifies observation of observable shadow latches and normal primary outputs.

- **Clockpo**

  A literal that specifies observation of only primary outputs directly connected to clocks.

Example

The following Tessent FastScan example specifies slave latches as the observation point for random pattern fault simulation:

```
set_system_mode analysis
set_observation_point slave
add_faults -all
simulate_patterns -source random
```

Related Commands

set_capture_clock
set_random_patterns
simulate_patterns

set_output_masks

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

Mode: setup

Usage

set_output_masks **OFf** | **ON** [-Bidi_exclude] [-Exclude *pin_pathname*…]

Description

Ignores any fault effects that propagate to the primary output pins you select.

The tool uses primary output pins as the observe points during the fault detection process. When you mask a primary output pin, you inform the tool to mark that pin as an invalid observation point during the fault detection process. This command allows you the ability to flag all or a portion of the primary output pins that do not have strobe capability (are not observable). The tool classifies the faults whose effects only propagate to that observation point as Atpg_Untestable (AU).

You can use the report_output_masks command to display the current setting of the output masks.

Arguments

- **OFf**

  A required literal that specifies for the tool to use primary output pins as the observe points during the fault detection process. In addition, the tool will give possible credit for fault effects that reach a pin that the pattern set specifies as being at an unknown value. This is the default behavior upon invocation.

- **ON**

  A required literal that specifies for the tool to mask the simulated primary output pin values with Xs for all patterns. As a result, the tool does not give any credit for fault effects that reach those masked primary output.

- -Bidi_exclude

  An optional switch that modifies how the command treats bidirectional pins.

Note

The meaning of the -Bidi_exclude switch is different for each setting (“off” or “on”) of the command. It basically specifies to do the opposite, for bidirectional pins, of what is done for other pins.

When used with the *Off* literal, the -Bidi_exclude switch specifies for the tool to set_output_masks “on” for bidirectional pins. Thus, the command “set_output_masks off -bidi_exclude” results in the tool masking bidirectional pins and no others.

Conversely, if you use the *On* literal, then adding the -Bidi_exclude switch causes the tool to set_output_masks “off” for bidirectional pins. The command “set_output_masks on -bidi_exclude”, therefore, results in the tool masking all pins except bidirectional pins.

- -Exclude *pin_pathname…*

  An optional switch and repeatable string that modifies how the command treats a specific pin.

Note

Similar to the -Bidi_exclude switch, the -Exclude switch specifies for the tool to do the opposite, for each specified pin, of what is done for other pins.

When used with the Off literal, the -Exclude switch will cause the tool to set_output_masks “on”, but only for the specified pin(s). The command “set_output_masks off -exclude my_pin”, for example, will result in the tool masking the pin whose pathname is “my_pin” and not masking any others.

Conversely, when used with the On literal, the -Exclude switch will cause the tool to set_output_masks “off” for the specified pin(s). In this case, “set_output_masks on -exclude my_pin” will apply masking to all pins except the pin “my_pin”.

Example

The following example reports all the primary outputs of a design having one primary output, /y_inout, that is bidirectional. It then specifies to mask all the primary outputs *except* any that are bidirectional, and the three pins, y_out[3], y_out[2], and x_out[1]. The example then displays the masked primary output pins. As you would expect, none of the pins excluded from masking are listed.

report_primary_outputs

```
SYSTEM:  /x_out[3]
SYSTEM:  /x_out[2]
SYSTEM:  /x_out[1]
SYSTEM:  /x_out[0]
SYSTEM:  /y_out[3]
SYSTEM:  /y_out[2]
SYSTEM:  /y_out[1]
SYSTEM:  /y_out[0]
SYSTEM:  /my_out
SYSTEM:  /y_inout
```

set_output_masks on -bidi_exclude -exclude y_out[3] y_out[2] x_out[1]

report_output_masks

```
TIEX   /x_out[3]
TIEX   /x_out[2]
TIEX   /x_out[0]
TIEX   /y_out[1]
TIEX   /y_out[0]
TIEX   /my_out
```

Related Commands

add_output_masks
delete_output_masks
report_environment
report_output_masks

set_parallel_load_subchains

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

Mode: setup, analysis

Usage

set_parallel_load_subchains [OFf | ON]

Description

Specifies whether the internal parallel access functionality is enabled or disabled.

By default, internal parallel access is disabled. In this case, when a netlist contains a subchain in a library model, the parallel testbench can only apply serial shifts to load the scan data and compare the capture values because internal pin names (where the parallel value is forced and observed) of the library models cannot be accessed. This limitation can increase parallel simulation runtime and significantly increase the size of the parallel testbench file.

If this command is enabled in Setup mode, the parallel testbench is written to access internal pin names of the library models. When you exit Setup mode, a DRC warning is issued if the netlist contains a subchain. In this case, you should enable this command. You can use the + command to display detailed information about the subchains in the netlist.

Arguments

- **OFf**

  An optional literal that disables internal parallel access. This is the default.

- **ON**

  An optional literal that enables internal parallel access. This command can be disabled or re-enabled in analysis mode to write out different parallel test benches if necessary, but only if it was initially enabled in Setup mode. If this command was not enabled in Setup mode, you cannot change the setting in analysis mode because internal access points can only be learned during DRC when switching from Setup mode to a non-Setup mode.

Note

When this command is enabled, extra memory is required for storing the internal names of the library cells. Also, internal pin names of the ATPG library models must match the internal pin names of the Verilog library model; otherwise, testbench compile errors or incorrect results from a Verilog simulation can occur. If you used LibComp to translate the Verilog libraries to ATPG libraries, then internal pin names should match because LibComp preserves these internal names.

Example

The following example enables internal parallel access in Setup mode, sets the system mode to ATPG, creates test patterns, and saves the test patterns with parallel loading of subchains.

```
set_parallel_load_subchains on
set_system_mode analysis
create_patterns
write_patterns trans_pat.gz -verilog -parallel
```

If the testbench is not compiled because the internal pin names of the Verilog library do not match those of the ATPG library, parallel loading can be disabled. Then the testbench loads the subchain cells serially.

```
set_parallel_load_subchains off
write_patterns trans_pat2.gz -verilog -parallel
```

Related Commands

report_environment

report_scan_chains

set_pathdelay_holdpi

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

Mode: setup, analysis

Usage

set_pathdelay_holdpi **OFf** | **ON**

Description

Specifies whether the ATPG keeps non-clock primary inputs at a constant state after the first force.

The set_pathdelay_holdpi command lets you specify for the tools to ignore non-clock primary input changes after the first force in each pattern.

Arguments

- **OFf**

  A literal specifying that the tool can change non-clock primary input values at any time. This is the default behavior upon invocation of the tool.

- **ON**

  A literal that specifies for the tool not to change non-clock primary input values after the launch of the transition into the path.

Example

The following example enables Tessent FastScan to ignore changes to non-clock primary inputs:

```
set_pathdelay_holdpi on
```

Related Commands

set_transition_holdpi

set_pattern_buffer

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

Mode: setup, analysis

Usage

set_pattern_buffer {***directory_name...***} | **-Off**

Description

Stores run-time pattern data in temporary files in the directory specified.

The set_pattern_buffer command allows you to specify a directory where the tool can store temporary run-time pattern data. When the tool accesses a pattern, it is then automatically retrieved from the specified directory, as opposed to virtual memory, conserving the memory for other tool processing tasks. This provides a run-time processing advantage when there is not ample swap space.

You can specify multiple directories. As individual buffer files approach their file size limit or volume space runs out, the tool searches the specified directory list for available space. The tool deletes these buffer files when you exit the application.

Note

Buffer files may be multiple gigabytes large. Choose directories on volumes that have enough disk space to handle expected amount of pattern data.

Arguments

- ***directory_name***

  A required repeatable string that specifies the full path directory names in which the tool saves buffer files, thus enabling the usage of temporary buffer files for pattern data. Re-issuing this command with additional directories causes the tool to add the directories to the current list of directories available for buffer files.

- **-Off**

  A required switch that disables the use of temporary buffer files, causing the tool to save run-time data in virtual memory. This is the default upon invocation. Disabling the usage of pattern buffers can only be done when there is no pattern data, such as after issuing the reset_state or set_system_mode commands.

Example

The following example places buffer files in the specified directories.

```
set_pattern_buffer /usr/jdoe/buffers /usr2/jdoe/morebuffers
```

Related Commands

report_resources

set_pattern_classification

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

Mode: setup, analysis

Usage

set_pattern_classification **ON** | **OFf**

Description

Specifies the order in which to store test patterns.

When set to “on,” each pattern is stored by its classification, according to the following ordered list:

basic
read only (a type of basic pattern)
ram sequential
clock sequential and multi-load

When set to “off,” test patterns are stored in the order in which they were generated. Note that when simulating the external patterns (by issuing the simulate_patterns command), the tool automatically turns off pattern classification.

Arguments

- **ON**

  A literal that enables pattern classification. If the current internal pattern set is not classified, the pattern is automatically classified after the command is executed.

- **OFf**

  A literal that turns off pattern classification. When pattern classification is disabled, patterns are stored in the same order in which they were generated.

Example

The following example stores patterns in the order in which the tool generates them:

```
set_pattern_classification off
```

Related Commands

write_patterns

set_ram_test

set_pattern_filtering

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Usage

set_pattern_filtering [-EXternal] -Off | -LIst ***pattern_index***…
| {{[{-PATtern_type ***keyword***}… | {-NOPAttern_type ***keyword***}…]
[-SAMple_per_type ***integer***] [-CAPture]}
[{-RANge ***begin_int end_int***}…] [{-NORange ***begin_int end_int***}…]
[{-NAmed_capture_procedures {**procedure_name**... | **ALL**}}
| {{-CLocks ***clock_name***…} {-NO_CLocks [***clock_name***…]}}
| {-RESTRICT_CLocks ***clock_name***…}]}
{[-CAPTURE_Power ***operator percentage***
[ST_Peak | ST_Average | WSA_Peak | WSA_Average]
[-INSTance ***object_name***...] [-MODUle ***module_name***...]}
{[-SHIFT_Power {LOAD | RESPonse} ***operator percentage***] [-INSTance ***object_name***...]
[-MODUle ***module_name***...]}

Description

Creates a temporary set of sampled scan patterns.

You can then save the sampled patterns in all supported formats with the write_patterns command. To restore the original non-sampled patterns, issue set_pattern_filtering -Off. If the content of the original pattern set changes while pattern filtering is on, pattern filtering is turned off automatically.

Note that the sampled patterns may be ordered differently than the original patterns. Also, the filtering and ordering only applies to scan (ATPG) patterns, and is not performed for chain test patterns.

Note

Because MacroTest patterns are order dependent, they are sampled sequentially, beginning at the first pattern. Non-MacroTest patterns are re-ordered.

Arguments

- -EXternal

  An optional switch that applies sampling to external patterns. Before using this switch, you must specify the name of the external pattern file with the read_patterns command. The tool also re-orders the patterns.

- -Off

  An optional switch that restores the original non-sampled patterns.

- -LIst *pattern_index…*

  An optional switch and repeatable, non-negative, integer pattern indexes that creates a sample consisting of patterns with the specified pattern indexes, and orders the sampled patterns in the same order the indexes appear (left to right) in the command. The *pattern_index* refers to original pattern indexes in the pattern set before filtering by the set_pattern_filtering command. This switch cannot be combined with any other pattern filtering options.

  This switch enables you to arrange patterns in any order. For example, during IDDQ test, if the current of every pattern is measured to sort them in order of increasing current, the result is a list of pattern indices (for example, 13, 2, 24, and so on) that does not match any of the tool's built-in ordering criteria. You could use this switch to order the patterns according to the IDDQ sort.

- -PATtern_type *keyword*

  An optional switch and keyword pair that applies sampling based on the *keyword* pattern type. Multiple "-Pattern_type *keyword*" pairs are allowed; however, you cannot combine this switch with the -Nopattern_type switch in the same command. Keywords are listed in Table 7-47. If you specify neither this switch nor the -Nopattern_type switch, the default is to sample all pattern types.

  Valid pattern type keywords are:

Table 7-47. Keywords for Pattern Types

Pattern Type	Keyword
Basic	BAsic
Clock PO	CLOCK_Po
Clock Sequential	CLOCK_Sequential
MacroTest	MacroTest
Multi-load	MUlti_load
RAM Sequential	RAm_sequential

- -NOPattern_type *keyword*

  An optional switch and keyword pair that specifies to not sample the *keyword* pattern type. Multiple "-Nopattern_type *keyword*" pairs are allowed; however, you cannot combine this switch with the -Pattern_type switch in the same command. Keywords are listed in Table 7-47.

- -SAMple_per_type *integer*

  An optional switch and positive integer pair that specifies how many patterns of each type to sample.

- -CAPture

  An optional switch that samples the patterns for a different capture clock, instead of a different pattern type.

  If you specify the -Capture switch without the -Sample_per_type switch, the command samples just one pattern for each capture clock. If you specify the -Capture switch with the -Pattern_ type switch, the command only samples patterns within the specified pattern types.

- -RANge *beg_int end_int*

  An optional, repeatable switch and pair of integers that creates a sample of patterns beginning with pattern *beg_int* and ending with pattern *end_int*. Patterns are numbered beginning with 0; for example, to restrict sampling to the first 100 patterns, use "-range 0 99."

- -NORange *beg_int end_int*

  An optional, repeatable switch and pair of integers that specifies to not apply sampling to the range of patterns beginning with pattern *beg_int* and ending with pattern *end_int*. Patterns are numbered beginning with 0; for example, to exclude patterns 51 through 60 from sampling, use "-norange 51 60." You can use this switch to exclude sub-ranges within sampled ranges specified with the -Range switch.

- -NAmed_capture_procedures *procedure_name*... | ALL

  An optional switch and repeatable string or literal pair that samples patterns for either the specified named capture procedures or all named capture procedures.

  The number of patterns sampled depends on the number of samples specified using the -sample switch. If the -sample switch is not used, all the patterns that use the specified named capture procedures are sampled.

- -CLocks *clock_name…*

  An optional switch and repeatable string pair that samples only patterns that use one or more of the specified clocks. This includes patterns that use unspecified clocks if they also use at least one of the specified clocks. For example, "set_pattern_filtering -clocks clk1 clk2 clk3" will sample a pattern that pulses clk1, clk2 and clk4. Patterns that do not pulse any clock are not sampled.

  You cannot combine this switch with the -Restrict_clocks switch in the same command.

- -NO_CLocks *clock_name…*

  An optional switch and optional, repeatable, string pair that samples only patterns that do not use any of the specified clocks. To obtain a sample that includes all the patterns that do not pulse any clock, omit the *clock_name* argument.

Note

If you issue “set_pattern_filtering -no_clocks clk1 clk2,” where clk1 and clk2 are the only clocks pulsed by any pattern, the filtering result is the same as omitting the clock names and simply issuing “set_pattern_filtering -no_clocks.”

You cannot combine this switch with the -Restrict_clocks switch in the same command. If you omit the *clock_name* argument, you cannot combine this switch with the -Clocks switch in the same command.

- -RESTRICT_CLocks *clock_name…*

  An optional switch and repeatable string pair that samples only patterns that use the specified clocks, or a subset of these clocks, exclusively. Patterns that use any unspecified clocks are not sampled, even if they also use some or all of the clocks specified. For example, “set_pattern_filtering -restrict_clocks clk1 clk2 clk3” will not sample a pattern that pulses clk1, clk2 and clk4. Patterns that do not pulse any clock are not sampled.

  You cannot combine this switch with the -Clocks or -No_clocks switch in the same command.

- -CAPTURE_Power

  Optional switch that filters test patterns based on switching activity during capture.

- ***operator***

  Optional literal that specifies an operation for test pattern filtering based on switching activity during shift or capture. Literal options include:

  < — Less than

  > — Greater than

  <= — Less than or equal to

  >= — Greater than or equal to

- ***percentage***

  Optional integer that specifies a switching threshold percentage that applies to filtering test patterns based on capture or shift power. The specified integer must be positive.

- ST_Peak | ST_Average | WSA_Peak | WSA_Average

  Optional literal that filters the test patterns based on a specified metric. These options can only be used for switching activity during capture. Literals include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  WSA_Average — Average of weighted switching activity.

- -INSTance ***instance_pathname***...

  An optional switch and repeatable string that specifies one or more target instance(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The instance_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -MODULe ***module_name***...

  An optional switch and repeatable string that specifies one or more target module(s) for test pattern sampling. This option is only available for sampling local switching activity during shift and/or capture. The module_pathname may include any number of asterisk (*) and/or question mark (?) wildcard characters in a name.

- -SHIFT_Power {LOAD | RESPonse}

  An optional switch and literal that filters test patterns based on switching activity during shift. Literal options include:

  LOAD — filters based on switching during test pattern loading.

  RESPonse — filters based on switching during the circuit response to test patterns.

Example 1

The following example assumes ATPG has just been run, resulting in 125 patterns of three different types, including clock PO. There are two capture clocks. The example first saves a sample consisting of four clock PO patterns to a separate pattern file in TITDL format. The example then samples two patterns for each pattern type. Next, the same command is repeated with the addition of -Capture. This causes the tool to sample two patterns per type as before, and then add as necessary, additional patterns to ensure the sampled pattern set contains at least two samples per capture clock.

report_pattern_filtering

```
// Pattern filtering is off.
```

set_pattern_filtering -pattern_type clock_po -sample 4

```
// Pattern filtering is on: 1 pattern type(s),
    4 selected pattern(s), 125 original pattern(s).
```

write_patterns clock_po.tdl -titdl -repl

```
// Note: pattern filtering is on, #-saved-pattern(s) = 4,
    #-total-pattern(s) = 125
```

set_pattern_filtering -sample 2

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
   6 selected pattern(s), 125 original pattern(s).
```

set_pattern_filtering -capture -sample 2

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 3 pattern type(s),
```

```
   7 selected pattern(s), 125 original pattern(s).
```

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 12
   sampled_patterns_per_type     = 2
   sample_capture_clock          = on
   sample_pattern_type(s)        = all
-----------------------------------------------
```

Example 2

The following example creates a sample of patterns 0 through 4, 6 through 10, and 20 through 29.

set_pattern_filtering -range 0 10 -norange 5 5 -range 20 29

```
// WARNING: The previous filter option has been reset.
// Internal pattern filtering is on: 20 selected pattern(s),
   125 original pattern(s).
```

report_pattern_filtering

```
   Current pattern filtering configuration
-----------------------------------------------
   original_patterns (internal)  = 125
   current_filtered_pattern(s)   = 20
   sampled_patterns_per_type     = all
   sample_capture_clock          = off
   sample_pattern_type(s)        = all
   include_range(s)              = [20, 29], [0, 10]
   exclude_range(s)              = [5, 5]
-----------------------------------------------
```

Example 3

The following example creates a sample of patterns that use one or more of the clocks clk1, clk2, and clk3 and other clocks except for clk4; patterns that use clk4 are not sampled.

set_pattern_filtering -clocks clk1 clk2 clk3 -no_clocks clk4

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 6 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//        pre-
//        filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      0   basic       1     1       -        -    [clk3]
//    1      3   basic       1     1       -        -    [clk1]
//    2      5   clk_seq     2     1       -        -    [clk1] [clk2]
//    3      6   clk_seq     2     1       -        -    [clk2] [clk3]
//    4      9   clk_seq     2     1       -        -    [clk2] [clk5]
```

```
//    5     18   clk_seq     2     1       -        -    [clk2] [clk6]
```

Notice that when pattern filtering is in effect, the report_patterns command displays the index numbers of the patterns prior to filtering as well as after filtering.

Example 4

The following example creates a sample of patterns that do not pulse any clock.

set_pattern_filtering -no_clocks

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 2 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//        pre-
//        filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      1   basic       1     1       -        -    [-]
//    1      2   basic       1     1       -        -    [-]
```

Example 5

The following example creates a sample of patterns 54, 106, 34 and 125, in that order.

set_pattern_filtering -list 54 106 34 125

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on: 4 selected pattern(s),
// 125 original pattern(s).
```

report_patterns

```
//        pre-
//       filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0     54   basic     1       1       -        -    [clk2,clk4]
//    1    106   clk_seq   3       1       -        -    [clk1] [clk3] [c
//    2     34   mult_ld   4       3       -        -    [clk1]
//    3    125   clk_seq   2       1       -        -    [clk1] [clk2]
```

Example 6

The following example creates a sample of patterns that use the specified named capture procedures TranPtrn01 and TranPtrn03.

set_pattern_filtering -named TranPtrn01 TranPtrn03

```
//  Warning: The previous filter options have been reset.
//  Internal pattern filtering is on: 17 selected pattern(s), 32 original
pattern(s).
```

report_patterns

```
//        pre-
```

```
//       filter
// patt # patt #   type   cycles loads obs_proc cap_proc capture_clk_seq
// ------ ------ -------- ------ ----- -------- -------- ---------------
//    0      0   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    1      1   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    2      2   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    3      3   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    4      4   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    5      5   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    6      6   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    7      7   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    8      8   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//    9      9   clk_seq   2       1       -  TranPtrn01 [clk] [clk]
//   10     19   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   11     20   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   12     21   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   13     22   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   14     23   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
//   16     25   clk_seq   2       1       -  TranPtrn03 [clk] [hclk]
```

report_pattern filtering

```
    Current pattern filtering configuration
-----------------------------------------------
    original_patterns (internal) = 32
    current_filtered_pattern(s)  = 17
    sampled_patterns_per_type    = all
    sample_capture_clock         = off
    selected named capture proc  = TranPtrn01, TranPtrn03
    sample_pattern_type(s)       = all
-----------------------------------------------
```

Example 7

The following example creates a sample of two patterns for each of the specified named capture procedures TranPtrn01 and TranPtrn03 and then saves the sampled patterns.

set_pattern_filtering -sample 2 -named TranPtrn01 TranPtrn03
write_patterns pat_TranPtrn01_TranPtrn03.ascii

Example 8

The following example creates a sample of two patterns for all the named capture procedures and then saves the sampled patterns.

set_pattern_filtering -sample 2 -named ALL
write_patterns pat_all.ascii

Example 9

The following example creates a sample of test patterns in which the peak weighted switching activity of the my_bypass_2_1_instance_1 instance is below the threshold 60%, and the peak WSA of the my_bypass_2_1_instance_2 instance is below the threshold 55%:

create_patterns
**set_pattern_filtering -capture_power WSA_PEAK -power_percentage < 60 -instance **
**cpu_top/p10/pic2/my_bypass_2_1_instance_1 -power_percentage_not_exceeding **
55 -instance cpu_top/p10/pic2/my_bypass_2_1_instance_2

```
// Warning: The previous filter options have been reset.
// Internal pattern filtering is on:  65 selected patterns,
// 83 original patterns.
```

Related Commands

order_patterns
report_environment
report_pattern_filtering
report_patterns
write_patterns
read_patterns

set_pattern_source

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

Mode: analysis

Usage

When EDT is Off

set_pattern_source **Internal** | {**Random** | **Bist** | {{**External** ***filename***}
[-APpend | -Replace] [-Binary | -AScii | -STil | -WGl]}
[-MAsk [*mask_filename* [-MASKFILE_pattern_offset *number*]]]
[-PREserve_masked_observe_points] [-NOPadding] [-Store_patterns] [-ALl_patterns]}

When EDT is On

set_pattern_source **Internal** | {{**External** ***filename***}
[-APpend | -Replace] [-Binary | -AScii | -STil | -WGl]
[-MAsk [*mask_filename* [-MASKFILE_pattern_offset *number*]]]
[-PREserve_masked_observe_points] [-Store_patterns] [-ALl_patterns]}

Description

Specifies the source of the test patterns.

Note

The set_pattern_source command followed by the “run” command has been mostly superseded by the read_patterns and simulate_patterns commands. For “set_pattern_source internal” and “run”, the replacement command is create_patterns. Mentor Graphics recommends you use the newer commands.

Arguments

- **Internal**

  A literal that specifies the internal set of patterns. This is the default upon invocation of the tool.

- **Random**

  A literal that specifies the random patterns, capture clock, and observation point specified previously.

- **Bist**

  A required literal that specifies BIST patterns.

  There must be at least one defined LFSR or an error displays. The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup.

- **External *filename***

  A literal and string pair that specifies the set of patterns contained in *filename* when performing a simulation run. For simulation, the specified patterns are loaded into an external pattern set database.

  In analysis system mode, the tool adds effective patterns from a simulation run to the internal pattern set, if there is at least one undetected fault in the fault list. When you use the -All_patterns switch, the tool adds all simulated patterns from the external pattern set to the internal pattern set, not just the effective patterns.

  In analysis mode, a simulation run does not alter the internal pattern set.

  The external patterns can be in ASCII, binary, STIL, or WGL test pattern format.

Note

You may notice a drop in test coverage when using an external pattern set as compared to using generated patterns. This is an artificial drop.

  There is an issue regarding artificial drops in test coverage due to redundant faults. When you run ATPG, the tool classifies some faults as redundant (RE). The tool knows these faults are untestable when calculating test coverage. If you restart the tool, or go back to Setup mode before reading the patterns back in, the faults that were previously classified as RE will now be classified as UC or UO. Now, the tool does not know these faults are untestable when calculating test coverage because faults cannot be proven redundant by fault simulation. The tool will only fault simulate the external patterns. It will not target undetected faults and try to prove them redundant as it does when the pattern source is internal. Therefore, you may see a slight drop in test coverage. This drop is an artificial drop and coverage still remains the same.

  The external pattern formats are described in the *Tessent Scan and ATPG User's Manual*.

- -APpend

  An optional switch that specifies for the tool to append the patterns contained in *filename* to the patterns in the tool's current external pattern set.

- -Replace

  An optional switch that specifies for the tool to replace its current external pattern set with the patterns contained in *filename*. This is the default.

- -Binary

  An optional switch that specifies the external test pattern set is in binary format. This is used when reading in a file saved with the write_patterns -BInary command.

  If neither -Binary or -Ascii options are specified, the tool tries to open and process the file as a binary file. If this is unsuccessful, the tool then tries to open and process the file as an ASCII file.

- -AScii

  An optional switch that specifies the external test pattern set is in ASCII format. You cannot use this option with the Internal pattern source.

- -WGl

  An optional switch specifying that the external test pattern set is in WGL format. You must use this option if you specify external patterns that are in WGL format.

  Restrictions:

  - For Tessent FastScan and Tessent TestKompress, you must load parallel scan patterns, not serial. Patterns produced by MacroTest cannot be read back in. For more information, see the note under the External argument.
  - If you re-simulate scan patterns, only serial patterns are allowed.

- -STil

  An optional switch specifying that the external test pattern set is in STIL format. You must use this option if you are loading STIL functional patterns into the tool.

  Restriction:

  - You must load parallel scan patterns, not serial.
  - The patterns must be STIL patterns originally created in these tools.

  Analysis of the STIL patterns conforms to the IEEE 1450.0 specification. The following features from the IEEE 1450.0 specification are not supported:

  - Multiple data elements per test cycle (See section 5.7 of IEEE 1450.0)
  - Non-cyclized test data (See section 5.9 of IEEE 1450.0)
  - Goto statement referring to a previous pattern label (See section 22.8 of IEEE 1450.0)
  - MatchLoop statement (See section 22.7 of IEEE 1450.0)

  Analysis of the STIL patterns only supports waveform descriptions in the WaveformTable that Mentor Graphics tools currently support. Therefore, a waveform description should not have more than two event edges, unless it is describing a double pulse. All other waveforms that have more than two event edges are not allowed. Also, timing expressions are limited to using only the +, -, *, /, (, and) operators on constants or variables that have time-based units.

- -MAsk

  An optional switch that masks certain observation points in external patterns added to the internal pattern set after simulation. This switch enables you to salvage patterns that result in unreliable failures on certain cells at the tester, or that fail verification due to simulation mismatches at a few observation points. Rather than discard failing patterns altogether and accept the accompanying significant reduction in test coverage, you can keep the patterns,

use this switch to mask the few problem observation points, and typically see only minimal impact on test coverage.

The tool obtains the masking information from a text file you create (see the *mask_filename* argument), or from the external patterns read in. In the case of external patterns, the tool applies masks wherever Xs appear in a pattern's unload and measure_po values.

Note

If you use the -Mask switch without specifying a maskfile, the tool cannot use multiprocessing.

This switch affects the capture values of only those patterns from the external pattern set that the tool has added to the internal pattern set after simulation. Therefore, if you write patterns using "write_patterns -external," you will not see the masking information from the mask file reflected in the saved patterns; the saved patterns will be exactly the same as the original patterns.

To save a pattern set that incorporates the masking information, use the run command to perform a simulation of the external pattern set you specified with "read_patterns -mask." Then use the write_patterns command *without* the -External switch to write the resulting internal pattern set to a file.

Tip: When you use the -Mask switch, the tool automatically executes the -Store_patterns switch with it; you do not need to include an explicit -Store_patterns in your command line.

- *mask_filename*

  An optional string used with the -Mask switch to specify a mask file. The mask file is a text file you create that lists the patterns and locations you want masked. Use the statements and syntax described in Table 7-48 to create a mask file. Use one statement per line. Precede comment text with a pair of forward slashes (//). An example mask file follows the table.

Note

Tessent TestKompress names scan chains internally. Depending on the design architecture, you must use this internal naming format to identify Tessent TestKompress scan chains in the mask file as follows:

Modular EDT — edt_<block_name>_channel<number>
Top-level EDT — edt_channel<number>

Use the report_edt_pins command to determine the number associated with a scan chain.

Note

The mask file must be in a pattern based format. Most ATE's return accumulated cycle-based failure data. Use the read_failures command to convert cycle-based failure data to pattern based data for the mask file.

Table 7-48. Mask File Statements

Statement	Usage Rules
<location>	Required. Specifies a location to mask or observe. Use as many location statements as you need. List all location statements that specify a pattern index in ascending order. Syntax format depends on the type statement that precedes the location statement, as follows: **EDT Off:** • If “type mask” or “type observe” precedes the location statement, use one of these two formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <chain_name> <chain_output_pin_name> **EDT On:** • If “type mask” or “type observe” precedes the location statement, use one of these three formats: <pattern_index> <primary_output_name> <pattern_index> <chain_name> <cell_ID> <pattern_index> <channel_name> <channel_cell_index> Specify the *channel_name* argument as follows: For non-modular flows: *edt_channel<number>*. For example, *edt_channel3*. For modular flows: *edt_<block_name>_channel<number>*. For example, *edt_block1_channel3*. Note, the *channel_cell_index* argument refers to the unload cycle. • If “type pin_mask” precedes the location statement, use one of the following three formats: <primary_output_name> <channel_name> <channel_output_pin_name>

Table 7-48. Mask File Statements (cont.)

Statement	Usage Rules
type <value>	Optional. Indicates if the locations specified by subsequent location statements are to be masked (forced to X) or observed. Specify one of the following values: • mask — specifies to mask the locations, for a specific pattern • observe — specifies to observe the locations and mask all others, for a specific pattern • pin_mask — specifies to mask the locations, for *all* patterns. Overrides other type statements that cover the same location. Use as many type statements as you like. By default, the tools assume "type mask" for locations listed before the first type statement. A type statement inserted between two location statements for the same pattern does not take effect until the next different pattern.

Following is an example mask file:

```
//file: my_pattern_mask.ascii
//
// The next 3 lines mask 3 locations in pattern 20.
20 primary_output_12
20 chain2 31
20 chain3 101
// The next line masks 1 location in pattern 25.
25 chain1 120
type observe
// The following 2 lines mask all but the 2 specified locations in
//   pattern 40.
40 chain5 3
40 primary_output_1
// The following type statement takes effect with pattern 61.
type mask
// The next line defines a third observe location for pattern 40.
40 primary_output_2
61 chain4 120
// The next line masks chain5 for all patterns, overriding previous
//   "type observe" for cell 3 of chain 5 for pattern 40, and
//   subsequent "type observe" for cell 99 of chain5 for pattern 65.
type pin_mask
chain5
type observe
// The following line masks all cells in chain5 except cell 99 in
//    pattern 65.
65 chain5 99
```

- -MASKFILE_pattern_offset *number*

  An optional switch and integer that specifies the pattern offset number for the mask file. By default (when no offset is specified), the pattern in the mask file starts with pattern 0. This switch is useful for reducing the number of pattern files that must be loaded when only a subset need to be masked.

- -PREserve_masked_observe_points

  An optional switch that allows you to preserve existing masked observation points (scan cell or primary output) in external patterns loaded into the tool. Even if known values are determined after simulation, the tool continues to mask those locations. You can optionally specify "-Mask maskfile" with additional observation points to be masked.

Note

When you use the -PREserve_masked_observe_points switch, the tool cannot use multiprocessing.

- -Store_patterns

  An optional switch that directs the tool to place patterns it simulates during the Good or Fault system mode into the internal pattern set. You can then use the write_patterns command to save these patterns to an external file.

  You cannot use this option with the Internal pattern source.

- -ALl_patterns

  An optional switch that, in analysis mode only, specifies for the tool to add all simulated patterns to the internal pattern set, provided there is at least one undetected fault in the fault list. The simulated patterns can be either randomly generated or can be from the specified external pattern file. Patterns that cause bus conflicts, which the tool ordinarily discards, are included when you use this switch. Without this switch, the tool adds only the effective patterns from a simulation run to the internal pattern set.

- -NOPadding

  An optional switch specifying that the source test pattern set contains ASCII patterns that are not padded for the scan load and unload data. For example, the source pattern set may be one that you wrote with the write_patterns command using its -NOPadding switch.

  You cannot use this option with the Internal pattern source.

Example 1

The following example performs fault simulation on an external pattern file:

```
set_system_mode analysis
set_pattern_source external file1
add_faults -all
simulate_patterns
report_statistics
```

Example 2

The following example fault grades transition faults using a set of path delay patterns. Assume path delay patterns generated with "set_fault_type path_delay -mask_nonobservation_points" are stored in the file, *path_patterns_mask.ascii*:

```
set_system_mode analysis
set_pattern_source external path_patterns_mask.ascii -mask
set_fault_type transition
add_faults -all
simulate_patterns
report_statistics
```

Example 3

The following example shows how to combine patterns from several different files into one pattern set, which you can save in any supported format. First, the example loads patterns from three separate pattern files into the tool's external pattern set. The patterns are in the same order in which they are loaded. The example then writes the contents of the external pattern set into a new file in the WGL format. Notice the patterns loaded are in mixed formats (ASCII and binary); also, that the tool automatically knows to use the GNU gzip utility when processing the input file with the ".gz" extension.

```
set_pattern_source external my_patterns1.ascii
set_pattern_source external my_patterns2.binary -append
set_pattern_source external my_patterns3.ascii.gz -append
write_patterns my_patterns_all.wgl -wgl -external
```

Example 4

The following commands create a new pattern set using the existing pattern set pat.ascii and keep the mask observation points plus the additional masked bit specified in maskfile my_mask:

```
set_pattern_source external external pat.ascii -store \
        -preserve_masked_observation_points -maskfile my_mask
create_patterns
write_patterns pat_new_masks.ascii
```

Example 5

The following example demonstrates how to specify the offset to speed up the pattern mask flow. Assume that a group of pattern files are created as follows:

Pat1.stil & Pat1.v contain pattern index 0 to 999
Pat2.stil & Pat2.v contain pattern index 1000 to 1999
Pat3.stil & Pat3.v contain pattern index 2000 to 2999
Pat4.stil & Pat4.v contain pattern index 3000 to 3999

After Verilog simulation, only pattern file Pat3.v fails simulation with the following mismatch bit in maskfile Pat3.fail:

```
// This File is simulation generated (Pat3.v)
//format pattern
//failure_buffer_limit_reached none
//data_reference cell
//pattern_id  chain/PO_name  cell_number  expected_value  simulated_value

scan_test
   2001          chain1          513            L              H
// reg132/Q
last_pattern_applied          2999
// failing_patterns= 1 simulated_patterns= 1000  simulation_time= 2600000;
failure_file_end
```

The following command allows you to load only the mismatch pattern file and mask the mismatch bits, then save the pattern file without mismatches:

```
set_pattern_source external Pat3.stil -stil -store \
        -mask Pat3.fail -maskfile_pattern_offset 2000
create_patterns
write_patterns Pat3_masked.v -verilog
write_patterns Pat3_masked.stil -stil
```

Related Commands

write_patterns
set_abort_limit
set_capture_clock
set_observation_point
set_random_atpg
set_checkpointing_options

set_pattern_type

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

Mode: setup, analysis

Usage

set_pattern_type [-RAM_sequential {OFf | ON}] [-MAX_SEQuential { OFf | *depth* }] [-SEQuential *depth*] [-MULtiple_load {OFf | ON}] [-CLOck_po {ON | OFf}]

Description

Specifies the type of pattern(s) used by ATPG simulation.

Note

Switch settings for this command are cumulative; that is, switches retain their settings over multiple set_pattern_type commands until you specify a different setting.

Tip: If you use ATPG Expert (create_patterns), you do not need to issue this command. ATPG Expert will, for most designs, determine the optimal settings for this command. You may get better results by leaving this command to ATPG Expert.

The set_pattern_type command determines the type of patterns the tool uses during ATPG simulation.

The -Sequential switch provides the ability to use clock sequential cells. For example, “-sequential 2” enables the tool to create clock sequential patterns with up to two clock pulses between scan loads. Try to use the smallest possible depth, as it affects both memory requirements and performance. You can use the report_sequential_fault_depth command to obtain estimates of the maximum test coverage possible with different sequential depth settings.

When you increase the sequential depth, the tool places all current ATPG untestable faults in the untested fault class. However, multiple-clock pattern compression can be improved by increasing the sequential depth. You cannot decrease the sequential depth when there are any active patterns. You cannot set the depth greater than one if a clock procedure contains a restore_bidis statement since the procedure cannot use sequential ATPG.

If you turn on the RAM sequential option to allow RAM sequential test patterns in the test pattern file, the following rules apply:

1. You may not use RAM sequential patterns with the transition fault type.
2. You may not use RAM sequential patterns with the set_atpg_compression command, but you may use them with the compress_patterns command.
3. You may not use the RAM sequential patterns if a clock procedure contains a restore_bidis statement, since the procedure cannot use sequential ATPG.

4. If you are using external patterns that contain RAM sequential patterns, you must set the pattern type to RAM sequential with the -Ram_sequential switch.
5. Only RAMs which are proven stable during the load/unload process will be allowed to hold values from one scan load to the next and are testable with RAM sequential patterns.
6. You cannot change the pattern type from RAM or clock sequential while there are any active patterns in the internal or external pattern sets.
7. If you change the pattern type from combinational (the default) to RAM sequential, the tool places all current atpg_untestable faults in the undetected_uncontrolled class where they are available for additional fault simulation and test generation.
8. You may use failure diagnosis for pattern sets which contain RAM sequential patterns.
9. If you use the report_gates command with gate reporting set to parallel_pattern, then for the last set of 32 simulated patterns, the tool displays the values for the 2 to 4 vectors of the ram_sequential patterns. If you selected a RAM gate, the Report Gate command also displays the internal RAM values.
10. If you use the write_patterns command to save RAM sequential patterns, the tool places the argument “ram_sequential” on the pattern statement of the ASCII saved patterns.
11. The tool places RAM sequential patterns at the end of the internal pattern set.
12. The fault simulator can detect faults on RAM data lines during RAM sequential patterns, but the test generator will not attempt to create a RAM sequential test for a data line fault. The tool assumes that it can always detect the faults with a non-RAM sequential pattern.
13. Even when you set the pattern type to RAM sequential, the test pattern generator always attempts to find a combinational test first. The test pattern generator only attempts a sequential test generation if all of the following are true:
    - The test pattern generator identified the fault as combinationally ATPG untestable during the combinational test.
    - The pattern type is set to RAM sequential.
    - The fault is connected to an address or write line of an eligible RAM.

    The test pattern generator then creates a sequential test depending on how the fault propagates to the RAM. The test pattern generator will only try to create a test that satisfies one of the following conditions, and if unsuccessful, it will consider the fault to be aborted even if the maximum number of remade decisions has not been exceeded:

    - Write Port Address Lines Faults:

      Vector 1 - For the first data line of the fault write port, write 0 into the address where the fault address line is at the fault value, and the other address lines are 0 (address A).

Vector 2 - For the first data line of the fault write port, write 1 into the address where the fault address line is at the complements of the fault value, and the other address lines are 0.

Vector 3 - From the first data line of the first read port, read 0 from address A.

- Read Port Address Line Faults:

  Vector 1 - For the first data line of the first write port, write 0 into the address where the fault address line is at the fault value, and the other address lines are 0.

  Vector 2 - For the first data line of the first write port, write 1 into the address where the fault address line is at the complement of the fault value, and the other address lines are 0 (address A).

  Vector 3 - From the first data line of the fault read port, read 1 from address A.

- Write Line Stuck-Off Faults on Multi-Write Port RAMs:

  Vector 1 - For the first data line of the first non-fault write port, write 0 into address 0.

  Vector 2 - For the first data line of the fault write port, write 1 into address 0

  Vector 3 - From the first data line of the first read port, read 1 from address 0.

- Write Line Stuck-On Faults:

  Vector 1 - For the first data line of the fault write port, write 1 into address 0.

  Vector 2 - For the first data line of the fault write port, write 0 into address 0.

  Vector 3 - From the first data line of the first read port, read 0 from address 0 with the first data line of the fault write port at 1.

Arguments

- -RAM_sequential OFf | ON

  An optional switch and argument pair that specifies whether or not the test pattern set contains RAM sequential test patterns. This type of pattern is detailed in the "RAM Sequential Patterns" section of the *Tessent Scan and ATPG User's Manual*. This switch is off upon invocation of the tool.

- -MAX_SEQuential OFf | *depth*

  An optional switch and argument pair that specifies the maximum clock sequential depth if the tool automatically increases the depth during test generation. (When using the create_patterns command to generate test patterns, the tool may increase sequential depth automatically in order to achieve the maximal test coverage.) The Off option (the default) disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number. When using this switch to set maximum clock sequential depth, the depth set by the -Sequential switch cannot be greater than the maximum clock sequential depth.

- -SEQuential *depth*

  An optional switch and integer pair that specifies the depth of the design's non-scan sequential elements (its sequential depth). The integer can be a value between 0 and 255, but the best practice is to set the depth as low as possible because it affects both memory requirements and performance. The default sequential depth upon invocation is 0.

  A setting of 0 enables the tool to create only non-sequential patterns, while a setting of 1 enables the tool also to create basic sequential patterns; with either setting resulting in patterns having a maximum sequential depth of 1. A setting of 2 or more enables clock sequential pattern generation.

Note

Clock sequential patterns must be avoided if a clock procedure contains a restore_bidis statement, as the procedure cannot use clock sequential ATPG.

- -MULtiple_load OFf | ON

  An optional switch and argument pair that allows the tool to create multiple load patterns when clock sequential pattern generation is enabled (the sequential depth is set to 2 more). This switch is an alternative to -Ram_Sequential for testing RAMs if the sequential depth is set to 3 or 4. Depth 4 is required if the RAM has data hold, H or H1, in its _cram_read port definition. Although advisable not to set depth higher than required, it is fine to just set depth to 4 if targeting RAMs with the -Sequential option. The multiple load pattern capability is off upon invocation of the tool.

  Refer to "Multiple Load Patterns" in the *Tessent Scan and ATPG User's Manual* for additional details about the multiple load pattern type.

- -CLOck_po ON | OFf

  An optional switch and argument pair that enables the tool to create clock PO patterns. For a detailed description of this pattern type, refer to "Clock PO Patterns" in the *Tessent Scan and ATPG User's Manual*. This switch is on by default.

Note

Clock PO patterns allow two timings for clock PIs, so an additional timeplate is required to apply these patterns on a tester.

Clock PO patterns cannot be generated if you have defined any non-floating, free-running clocks. For more information, see the add_clocks command.

Example 1

The following example displays the current pattern type settings:

set_pattern_type

```
//  Settings : -RAM_sequential OFF
//             -SEQuential     0
//             -MAX_SEQuential 5
```

```
//              -MULtiple_load  OFF
//              -CLOck_po       ON
```

Example 2

The following example disables the restriction for clock sequential depth so that the tool can increase the clock sequential depth to any number:

set_pattern_type -max_sequential off

```
//  Settings : -RAM_sequential OFF
//             -SEQuential     0
//             -MULtiple_load  OFF
//             -CLOck_po       ON
```

Example 3

The following example creates patterns with up to two clock pulses between scan loads:

set_pattern_type -sequential 2

```
//  Settings : -RAM_sequential OFF
//             -SEQuential     2
//             -MULtiple_load  OFF
//             -CLOck_po       ON
```

Example 4

The following example enables RAM sequential pattern generation, resets sequential depth to its default setting (0), and disables creation of clock PO patterns.

set_pattern_type -ram_sequential on -sequential 0 -clock_po off

```
//  Settings : -RAM_sequential ON
//             -SEQuential     4/0
//             -MULtiple_load  ON/OFF
//             -CLOck_po       OFF
```

For -SEQuential, the first number means that the clock sequential depth is 4 and the second number means that the maximal clock sequential depth for clock sequential test cube generation is 0.

For -MULtiple_load, the “ON” means that the clock sequential simulation engine allows to simulate test patterns with multiple loads, and “OFF” means that the multiple loads for clock sequential test cube generation is disabled.

Example 5

The following example specifies to create multiple load patterns for testing RAMs. Assuming RAM sequential pattern capability was enabled with a previous set_pattern_type command, the example turns off that capability (recall that the switches retain their settings over multiple commands—until you change them).

set_pattern_type -ram_sequential off -sequential 4 -multiple_load on

```
//  Settings : -RAM_sequential OFF
//             -SEQuential     4
//             -MULtiple_load  ON
//             -CLOck_po       OFF
```

Related Commands

report_sequential_fault_depth

set_physical_translation

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

Mode: setup, analysis

Usage

set_physical_translation [-Path_divider *path_divider*] [-Substitute *substitute*] [-Replacement *replacement*] [-Global_prefix *global_prefix*] [-Hier_prefix *hier_prefix*] [-STrip_prefix *strip_prefix*] [-Array_delimiters *array_delim*]

Description

Translates special Verilog characters to SPICE-readable characters Calibre applications can read.

You can restore any of these translation settings to its default setting by issuing the command without the corresponding argument. For specifics for when you use this command, see "Net Pair Identification with Calibre for Bridge Fault Test Patterns" in the *Tessent Scan and ATPG User's Manual* for complete information.

Arguments

- -Path_divider *path_divider*

  An optional switch and string specifying the instance path divider. The default is "/X".

- -Substitute *substitute*

  An optional switch and string for substituting a "#" (pound sign) for a leading character or set of characters for pins, wires, and instance names. By default, the tool substitutes leading "$"s (dollar signs) with "#"s (pound signs). You can specify a maximum of 10 characters for substitution. If you specify the -Replacement switch and string, then the tool uses this character for the substitution character.

- -Replacement *replacement*

  An optional switch and string replacing the default substitution character. The default is a "#" (pound sign). If you specify a character, then the tool uses this character instead of the default when substituting a leading character or characters for pins, wires, and instance names.

- -Global_prefix *global_prefix*

  An optional switch and string for changing the default global prefix character the tool inserts for pins, wires, and instance names. The default is a "#" (pound sign).

- -Hier_prefix *hier_prefix*

  An optional switch and string changing the hierarchy prefix. The default is " " (space).

- -STrip_prefix *strip_prefix*

  An optional switch and string stripping the prefix. This argument is in Calibre form (for example, to strip Xbogus_top/Xmodule/net, you specify -strip_prefix Xbogus_top).

- -Array_delimiters *array_delim*

An optional switch and string changing the array delimiter. The default is “[ ]” (square brackets).

set_possible_credit

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

Mode: setup, analysis

Usage

set_possible_credit ***percentage***

Description

Specifies the percentage of credit that the tool assigns possible-detected faults.

The **set_possible_credit** command specifies the percentage of possible-detected faults that the tool considers as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. For the equations that the tool uses in these calculations, refer to "Testability Calculations" in the *Tessent Scan and ATPG User's Manual*.

When you invoke the tool, the default credit value for possible-detected faults is 50 percent.

Arguments

- ***percentage***

  A required integer, from 0 to 100, that specifies the percentage of possible-detected faults that you want the tool to consider as detected when calculating the test coverage, fault coverage, and ATPG effectiveness. The default value upon invocation of the tool is 50 percent.

Example

The following example sets the credit for possible detected faults as 25 percent for determining the fault coverage, test coverage, and ATPG effectiveness:

```
set_system_mode analysis
set_possible_credit 25
create_patterns
report_statistics
```

set_power_control

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

Mode: setup, analysis

Usage

Specify low-power shift and capture

set_power_control {OFf | ON} [-SWitching_threshold_percentage *percent_integer*]

Specify low-power shift

set_power_control SHIFT {OFf | ON}
[-SWitching_threshold_percentage *percent_integer*]
[-REjection_threshold_percentage {*percent_integer* | OFf}]

Specify low-power capture

set_power_control CAPTURE {OFf |ON }
[-SWitching_threshold_percentage *percent_integer*]
[-REjection_threshold_percentage {*percent_integer* [ST_Peak | ST_Average | WSA_Peak | WSA_Average] | OFf}]

Hybrid TK/LBIST Flow Usage

set_power_control **shift** {**off** | **on**} [-switching_threshold_percentage ***percent_integer***]

Description

Enables low-power ATPG and sets up a switching and/or capture threshold.

When the command is entered without any arguments, the current power control settings display.

Note

When the set_edt_power_controller command is set to *Enabled*, the default for the set_power_control SHIFT command is ON for TestKompress only.

For more information on using low-power ATPG with Tessent TestKompress, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

Hybrid TK/LBIST Flow Specifics

Specifies the switching threshold in the patterns for fault simulation.

You can only use the low-power shift usage of this command for LogicBIST, otherwise, the tool issues an error.

Note

You cannot use the command's existing -rejection_threshold_percentage switch with the low-power logicBIST controller.

Arguments

- CAPTURE OFf | ON

  Optional literal and switch that enables low-power capture. Options include:

  OFf — Disables low-power capture. Default.

  ON — Enables low-power capture.

- SHIFT OFf | ON
  Optional literal and switch that enables low-power shift. Options include:

  OFf — Disables low-power shift. Default.

  If the set_edt_power_controller command is set to *Enabled*, the *Off* option of set_power_control shift sets all masking bits to 1, which enables all internal chains; however, it does not disable low-power TestKompress hardware. To disable low-power TestKompress hardware and bypass the low power registers, you must force the edt_low_power_shift_en signal off in the test procedure file and execute the set_edt_power_controller *Disabled* command and option.

  ON — Enables low-power shift.

- -SWitching_threshold_percentage *percent_integer*

  Optional switch and integer pair that specifies the switching threshold for low-power ATPG. The percentage integer can be 1 to 100. By default, for Tessent TestKompress applications, the switching threshold for ATPG is set to match the maximum switching threshold defined for the power controller hardware if that threshold is equal to or greater than 15%; if maximum hardware switching is less than 15%, the software default is 15%. Note, maximum hardware switching is specified by set_edt_power_controller *Shift -MIN_Switching_threshold_percentage* ***percentage_integer***.

  For more information, see "Low-Power Test" in the *Tessent TestKompress User's Manual*.

- -switching_threshold_percentage ***percent_integer*** **(Hybrid TK/LBIST Flow Only)**

  An optional switch and integer pair that specifies the maximum switching activity during shift. The ***percent_integer*** value can be between 1 and 50.

  The tool runs fault simulation with the appropriate control register values to implement the specified switching threshold.

  When saving the tester format patterns in Tessent Shell, the registers are seeded with the same values used during pattern simulation. It is not possible to save hardware default mode tester patterns when the switching threshold specified during fault simulation is different from the switching threshold specified during IP creation. An error message will be issued if this switching threshold cannot be supported by the low-power hardware.

  Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

- [-REjection_threshold_percentage {***percent_integer*** | OFf}]

  Optional switch and integer pair that specifies a state element switching limit per test pattern. Test patterns that exceed the specified switching limit are discarded. By default, this option is off and if rejecting/discarding test patterns that exceed the threshold has a negative effect on test coverage, the specified threshold is violated.

- [ST_Peak | ST_Average | WSA_Peak | WSA_Average]

  Optional literal that filters the test pattern rejection threshold based on a specified metric. When a specified metric exceeds the rejection threshold, the pattern is discarded. These options can only be used with the low-power capture feature. Literals include:

  ST_Peak — Peak of state transitions. Default.

  ST_Average — Average of state transitions.

  WSA_Peak — Peak of weighted switching activity.

  WSA_Average — Average of weighted switching activity.

Example 1

The following example sets low-power capture with a 30% threshold and low-power shift with a 20% switching threshold and generates test patterns:

```
set_power_control capture on -switching_threshold_percentage 30 \
        -rejection_threshold_percentage 35
set_power_control shift on -switching_threshold_percentage 20 \
        -rejection_threshold_percentage 25
create_patterns
```

Example 2

The following example creates compression logic with a power controller configured to accommodate a minimum of 20% switching threshold and enabled for test pattern generation.

```
set_edt_power_controller shift -min_switching_threshold_percentage 20
…
write_edt_files low_power_enabled_edt.v
```

Given the compression logic created with the preceding command, the following commands are used during test pattern generation to enable the controller with a 22% switching threshold for shift and generate test patterns.

```
set_power_control shift on –switching_threshold_percentage 22
set_system_mode analysis
create_patterns
```

Example 3 (Hybrid TK/LBIST Flow Only)

This example configures the low-power LogicBIST logic to target a switching threshold of 10%. The actual values loaded in the low-power control registers to achieve this switching threshold can be seen using the report_edt_configuration command. The power metrics can be seen using the report_power_metrics command.

```
set_power_control shift on -switching 15
```

set_system_mode analysis

report_edt_configuration

```
//
//  IP version:                     5
//  Shift cycles:                   128,  120 (internal scan length) + 8
//                                  (additional cycles)
//  External scan channels:         4
//  Internal scan chains:           480
//  Masking bits:                   53
//  Longest chain range:            2 - 120
//  Decompressor size:              32
//  Compactor type:                 Xpress
//  Scan cells:                     57600
//  Compression per pattern:        112.50x (ATPG bypass = 4 x 14400,
//                                  EDT = 4 x 128)
//  Clocking:                       edge-sensitive
//  Compactor pipelining:           Off
//  Chain mask:                     On (load 0, unload 0)
//  LBIST:                          On
//    MISR compactor pipelines:     0
//    PRPG seed:                    00000001
//    Low power shift controller:   Enabled and active
//      Min switching threshold:    15%
//      Hold value:                 1010
//      Toggle value:               1011
//      Switching control:          0011
//
```

report_power_metrics -shift

```
//  Note: Pattern classification is automatically turned off.
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
Load Shift Transitions       4.45%   14.26%  23.51%
Response Shift Transitions  42.48%   49.16%  50.04%
```

Related Commands

report_edt_configurations
set_edt_power_controller
report_power_metrics

set_power_domain

Context: dft -scan

Mode: setup, analysis

Usage

set_power_domain {{***Power_domain_name…*** | **-All**}
[-Number_of_chains *integer* | -Max_length *integer*] }

Description

Specifies the number or the length of scan chains to be generated for the given power domains.

Arguments

- ***Power_domain_name…*** | **-All**

  A required repeatable string or switch that specifies the power domains to which the values of the -Number_of_chains or -Max_length arguments are applied.

- -Number_of_chains *integer*

  An optional switch and integer pair that specify the exact number of core scan chains to be inserted for each specified power domain. The default number of chains per power domain is 1.

- -Max_length *integer*

  An optional switch and integer pair that specify the maximum number of scan cells to be stitched into a scan chain of the specified power domains. The tool evenly divides the scan cells of a power domain into scan chains that are smaller than the -Max_length value.

Example1

The following example specifies to add two core scan chains to each power domain.

```
set_power_domain -all -number_of_chains 2
```

Example 2

The following example specifies to that each scan chain can contain no more than 30 scan cells.

```
set_power_domain -all -max_length 30
```

Related Commands

report_scan_partitions

set_power_metrics

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

Mode: setup, analysis

Usage

set_power_metrics [-SHIft {OFf | ON}] [-CAPture {OFf | ON}]
[-COMPosite_shift_power {OFf | ON}] [-X_CAPture_transition_credit *float*]
[-X_SHIft_transition_credit *float*]

Description

Calculates shift and capture power during test pattern generation and simulation.

Shift power is the test power consumed during the loading of test stimuli and the unloading of responses. Capture power is the test power consumed by switching activity in the circuit during capture cycles.

Arguments

- -SHIft OFf | ON

  An optional switch and literal pair that enables the calculation of the power consumed for switching activity in the circuit during shift.

- -CAPture OFf | ON

  An optional switch and literal pair that enables calculation of the power consumed for switching activity in the circuit during capture cycles.

- -COMPosite_shift_power OFf | ON

  An optional switch and literal pair that combines the load and unload switching activity in shift power calculations. By default, the load and unload switching activity is computed separately.

- -X_CAPture_transition_credit *float*

  An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x capture transitions when calculating switching activity in capture cycles (the default values are 0.1, 0.05, 0.05, respectively). The specified float value must be between 0 and 1, and it applies to all three x values.

- -X_SHIft_transition_credit *float*

  An optional switch and floating number pair that specifies the credit given for x-to-x, x-to-known, and known-to-x shift transitions when calculating switching activity during shift. The specified value applies to all x values. The float number should be between 0 and 1. The default is 0.5.

Example 1

The following example calculates shift power and capture power during test pattern generation, fault simulation, and good simulation and displays the capture power for capture cycles and the shift power for shift transitions.

set_power_metrics -shift on -capture on
create_patterns

```
            Statistics report
-------------------------------------------
                          #faults   #faults
fault class               (coll.)   (total)
-----------------------   -------   -------
FU (full)                  70256     88262
-----------------------   -------   -------
UC (uncontrolled)             92       118
UO (unobserved)              482       626
DS (det_simulation)        35006     44416
DI (det_implication)       17321     22589
PU (posdet_untestable)        46        62
PT (posdet_testable)          11        23
UU (unused)                  134       134
TI (tied)                     94        98
RE (redundant)               240       265
AU (atpg_untestable)       16830     19931
-----------------------   -------   -------
test_coverage              75.02%    76.39%
fault_coverage             74.52%    75.96%
atpg_effectiveness         99.17%    99.14%
-------------------------------------------
#test_patterns                          494
    #clock_sequential_patterns          494
#simulated_patterns                     512
CPU_time (secs)                        32.4
-------------------------------------------
Power Metrics                Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                          9.03%   18.74%  36.82%
State Element Transitions    8.99%   25.44%  47.33%
--------------------------  ------  -------  ------
Load Shift Transitions      39.06%   44.62%  51.66%
Response Shift Transitions   6.45%   39.38%  50.54%
-------------------------------------------
```

For Power Metrics:

- The WSA (Weighted Switching Activity) lists the minimum, average, and maximum percentage of the test power consumed by the switching activity of the combinational logic and state elements during capture cycles.
- The State Element Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of state element transitions during capture cycles.

- The Load Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of scan cell transitions during the loading of test stimuli.
- The Response Shift Transitions row lists the minimum, average, and maximum percentage of test power consumed in terms of the scan cell transitions during the unloading of responses.

Example 2

The following example reports the power metrics for external test patterns, *mypat.ascii*, with the composite shift power metric on and then off:

read_patterns mypat.ascii
set_power_metrics -composite on
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
Composite Shift Transitions 13.24%   20.08%  26.69%
```

set_power_metrics -composite off
report_power_metrics -external

```
Power Metrics                 Min.   Average   Max.
--------------------------  ------  -------  ------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Peak Cycle
--------------------------
WSA                         10.62%   12.54%  13.65%
State Element Transitions   13.59%   18.71%  20.94%
--------------------------  ------  -------  ------
Load Shift Transitions      15.38%   25.00%  30.77%
Response Shift Transitions  17.95%   21.15%  25.64%
```

Example 3

The following example sets up power metric calculations to use composite shift, generates test patterns, and reports shift power for all the patterns:

set_power_metrics -composite_shift on
create_patterns
report_power_metrics -shift -patterns all

```
//
//           Pattern            Shift
//  Pat. loaded Pat. unloadedAverage
//  ----------- -------------   -------
//  0            -              10.60%
//  1            0               8.60%
//  2            1              14.20%
//  3            2               8.20%
//  pad          3               9.56%
```

Example 4

The following example reports the power metrics with composite shift for both scan tests and chain tests:

set_power_metrics -composite_shift on
report_power_metrics -shift -all_test -patterns all

```
// Chain test :
//
//           Pattern          Shift
//  Pat. loaded Pat. unloadedAverage
//  ----------- -------------   -------
//  0           -               32.89%
//  1           0               48.60%
//  2           1              54.20%
//  pad         2              49.56%
//
// Scan test :
//
//           Pattern          Shift
//  Pat. loaded Pat. unloadedAverage
//  ----------- -------------   -------
//  0            -              10.60%
//  1            0               8.60%
//  2            1              14.20%
//  3            2               8.20%
//  pad          3               9.56%
```

Related Command

report_power_metrics

set_power_control

set_procedure_cycle_checking

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

Mode: setup

Usage

set_procedure_cycle_checking **ON** | **OFf**

Description

Enables test procedure cycle timing checking to be done immediately following scan chain tracing during design rules checking.

This command helps detect timing problems in test procedures earlier on in the ATPG process. By default, the test procedure cycle timing checking is performed after scan chain tracing. If an error condition is detected, the tool remains in the Setup mode. You can then modify the test procedures and reissue the set_system_mode command.

Arguments

- **ON**

  A literal that specifies for the tool to set_procedure_cycle_checking ON. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that specifies for the tool to set_procedure_cycle_checking OFF. In order to turn procedure cycle_checking off, you must be in Setup mode.

Example

```
set_system_mode setup
set_procedure_cycle_checking OFf
```

Related Commands

set_system_mode

set_procedure_retargeting_options

Context: patterns -scan, patterns -scan_retargeting

Mode: setup

Usage

set_procedure_retargeting_options [-scan {ON | OFf}]
[-ijtag {ON | OFf}]
[-timeplate *timeplate_name*]
[-pulse_during_shift *clock_port_list*]

Description

Controls the behavior of the automatic merging and mapping (retargeting) of core-level load_unload and shift procedures to top-level test procedures and mapping of core-level test_setup and test_end IJTAG procedures.

By default, the tool automatically retargets core-level test_setup and test_end IJTAG procedures, and adds the needed iCalls to the test_setup or test_end procedure. If you disable the -ijtag behavior by setting this switch to off, you will need to manually add all core level iCalls.

In patterns -scan context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level and merges the retargeted core-level procedures with top-level procedures if top-level procedures exist.

In patterns -scan_retargeting context, by default, the tool automatically retargets the core-level load_unload and shift test procedures to the chip-level if chip-level load_unload and shift test procedures do not exist.

If you disable the -scan behavior by setting it to off, and the chip-level load_unload and shift test procedures are missing from the top-level test procedure, the tool generates an error.

Arguments

- -scan {ON | OFf}

  A literal that specifies to the tool whether to enable test procedure retargeting. By default upon invocation, test procedure retargeting is enabled. In order to disable test procedure retargeting, you must be in Setup mode. For more information on test procedure retargeting, see "Core Mapping for ATPG Process Overview" in the *Tessent Scan and ATPG User's Manual*.

- -ijtag {On | Off}

  An optional switch that specifies whether automatic mapping of core-level IJTAG test_setup and test_end procedures to the next higher level of hierarchy is enabled. You should use the Off switch to disable automatic IJTAG mapping if you are manually generating the entire top-level test_setup and test_end procedures.

- -timeplate *timeplate_name*

  An optional switch that specifies the timeplate to be used in the generated load_unload and shift procedures. The tool determines the timeplate to use as follows:

  a. Uses the timeplate specified by the -timeplate switch if one is specified.

  b. If one is not specified:

     - In patterns -scan context, if the top-level load_unload and shift procedures are present, the tool uses the timeplate from the top-level procedure.

     - In patterns -scan_retargeting context or if top-level procedures are not provided, the tool uses the default test_setup timeplate.

  c. If neither a tool-generated or user-defined test_setup procedure exists, the tool uses the default timeplate. For more information on the tool-generated test_setup procedure, see the description in "Test Procedure Retargeting for Scan Pattern Retargeting" in the *Tessent Scan and ATPG User's Manual*.

  d. If the default timeplate is not specified, the tools uses the first timeplate read in.

- -pulse_during_shift *clock_port_list*

  An optional switch that specifies to pulse the clocks specified by the clock_port_list argument during the shift procedure. The clock_port_list argument can be a Tcl list of clocks or gate_pins that are primary inputs. To reset this specification, re-invoke this option and provide an empty list.

  This option is typically used to specify clocks that drive top-level pipeline stages outside the core. The tool can map the load_unload and shift procedures from the core(s) to the top. However, if top-level pipeline stages are clocked by separate clocks, the tool will not know to pulse them, and, as a result, will be unable to trace the scan channels from the cores to the top (R2 DRC violations). This option instructs the tool to additionally pulse the specified clock(s) in the created top-level shift procedure.

Example 1

The following example specifies a timeplate to use in the load_unload and shift procedures.

```
set_procedure_retargeting_options -timeplate gen_tp1
```

Example 2

The following example specifies to pulse two clocks.

```
set_procedure_retargeting_options -pulse_during_shift {top_clock pipe_clock}
```

Related Commands

set_procfile_name

set_procfile_name

Context: all contexts

Mode: setup

Usage

set_procfile_name **procedure_file_name**

Description

Specifies a new procedure file for the tool to process at a later time.

The set_procfile_name command reads the specified procedure file and does basic syntax checking. The tool reports any errors it finds but does not actually parse and store the file at the time you issue the command.

You can specify only one procedure file with this command, and if you issue the command again, the new procedure file overrides the one previously set.

If this command is used to specify a procedure file and scan group(s) where also added with procedure files, the scan group procedure files are parsed first, followed by this procedure file. Any procedure or timeplates specified in this procedure file replaces those defined in previous procedure files.

For scan pattern retargeting, specifies the top-level test procedure file to be used for saving top-level patterns. In this case, the following procedures are expected to be defined in this file: test_setup, load_unload, shift, and external_capture.

For core mapping for ATPG and pattern retargeting, you must provide the chip-level test procedure file; the test procedure file must minimally include the timeplate definition. You do not need to provide chip-level load_unload and shift test procedures; if they do not exist, the tool automatically generates them based on the information in the core-level test procedure files. This step merges the core test procedures if there is more than one core, and maps them to the top-level test procedures. You may also need to specify the test_setup procedure for the design, if you need to set up the chip. All other procedures will be automatically mapped by the tool if they are not specified.

Arguments

- ***procedure_file_name***

  A required string that specifies the pathname to the procedure file to be read.

Example 1

The following is an example of how this is used in an IJTAG retargeting flow.

```
set_context patterns -ijtag -no_rtl
...
set_procfile_name data/setup.proc # initial syntax checking done
set_system_mode analysis # procedure file opened and parsed here
```

Example 2

The "Retargeting Example" in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Commands

None

set_ram_initialization

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

Mode: setup

Usage

set_ram_initialization **Uninitialized** | **Random**

Description

Specifies whether to initialize RAM and ROM gates that do not have initialization files.

The set_ram_initialization command allows the tool to internally generate random values and place them into all uninitialized RAM and ROM gates. This command is useful when simulating random patterns.

Arguments

- **Uninitialized**

  A literal that specifies for the tool to use unknown (X) values to set the memory elements of all RAM and ROM gates which do not have an initialization file. This is the default behavior upon invocation of Tessent FastScan and Tessent TestKompress.

- **Random**

  A literal that specifies for the tool to use random values to set the memory elements of all RAM and ROM gates which do not have an initialization file.

Example

The following example places random values into all uninitialized RAM and ROM gates:

```
set_ram_initialization random
set_system_mode analysis
create_patterns
```

Related Commands

read_modelfile

write_modelfile

set_ram_test

Context: dft -edt, patterns -scan

Mode: analysis

Usage

set_ram_test **Static_pass_thru** | **Read_only** | **Pass_thru**

Description

Specifies the mode for RAM testing with random patterns.

The set_ram_test command specifies how Tessent FastScan tests RAM using random patterns. The default upon invocation is Static_pass_thru. However, if during the design rules check the tool encounters a violation that prevents you from using one of the RAM test modes, it displays a message telling you so and resets the RAM test mode accordingly. You can use the report_environment command to display the current RAM test mode.

The set_ram_test command does not affect the ATPG; it always considers all usable modes when creating a test pattern for a fault.

Arguments

- **Static_pass_thru**

  A literal that specifies to test RAMs in static-pass-through mode during random patterns. This is the default behavior upon invocation of Tessent FastScan.

- **Read_only**

  A literal that specifies to test RAMS in read-only mode during random patterns.

- **Pass_thru**

  A literal that specifies to test RAMs in dynamic-pass-through mode during random patterns.

Example

The following example exercises the RAM in pass-through mode for test generation:

```
set_system_mode analysis
set_ram_test pass_thru
create_patterns
```

set_random_atpg

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

Mode: setup, analysis

Usage

set_random_atpg **ON** | **OFf**

Description

Specifies whether the tool uses random patterns during ATPG.

The set_random_atpg command controls whether the tool uses random test generation techniques to create_patterns during the ATPG process.

Arguments

- **ON**

  A literal that specifies for the tool to use random patterns to create test patterns. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that specifies for the tool not to use random patterns to create test patterns. When you use this option, the tool only performs deterministic ATPG.

Example

The following example turns off the random ATPG process, so only the deterministic ATPG is performed:

```
set_system_mode analysis
set_random_atpg off
create_patterns
```

set_random_clocks

Context: dft -edt, patterns -scan

Mode: analysis

Usage

set_random_clocks {***pin_name*** | ***pin_pathname***}…

Description

Specifies whether Tessent FastScan uses combinational or clock_sequential patterns for random pattern simulation.

The set_random_clocks command specifies for Tessent FastScan to either use combinational random patterns or use the pins that you specify for clock_sequential random patterns during simulation.

Combinational random pattern simulation is the default upon invocation. You specify clock_sequential by entering the set_random_clocks command with an ordered set of clock/read/write lines which Tessent FastScan exercises in the same order during the clock_sequential random pattern simulation.

You can reset Tessent FastScan to its invocation default of combinational random pattern simulation by entering the set_random_clocks command without any arguments. Also, when you re-enter the Setup system mode, Tessent FastScan deletes the random clock list and is once again ready for combinational random pattern simulation.

Arguments

- ***pin_name*** | ***pin_pathname***

  A required repeatable string that specifies the names of defined clock, read, and write lines. You must list these pins in the order in which you want Tessent FastScan to exercise them during the clock_sequential random pattern simulation. The default is none, which specifies for Tessent FastScan to perform combinational random pattern simulation.

  Tessent FastScan displays an error if any specified pin is not a defined clock, read, or write line. The tool also displays an error if the number of pins that you list is equal to or exceeds the selected sequential depth.

Example

The following example runs random pattern simulation with clock_sequential patterns:

```
set_pattern_type -sequential 10
add_scan_groups g1 seqproc.g1
add_scan_chains c1 g1 si so
add_clocks 0 sk1 sk2
set_system_mode analysis
set_random_clocks sk1
add_faults -all
simulate_patterns -source random
```

Related Commands

set_capture_clock

simulate_patterns

set_random_patterns

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

Mode: setup, analysis

Usage

set_random_patterns ***integer***

Description

Specifies the number of random patterns that the tool simulates.

The set_random_patterns command specifies how many random patterns you want the tool to simulate.

Arguments

- ***integer***

  A required integer, greater than or equal to 0, that specifies the number of random patterns you want the tool to simulate. The invocation default is 1024.

Example

The following example sets the number of random patterns to analyze its controllability effects:

```
set_system_mode analysis
set_random_patterns 612
```

Related Commands

set_capture_clock
set_observation_point
simulate_patterns

set_register_value

Context: all contexts

Mode: all modes

Usage

set_register_value ***value_name value_string***
[-RAdix { BInary | HEx | OCtal | Decimal }]

Description

Provides a method for specifying the register value variable outside of Setup mode.

When using this command, you must have previously specified the add_register_value command with the -Width command as well as the required arguments.

See "Serial Register Load and Unload for LogicBIST and ATPG" in the *Tessent Shell User's Manual* for complete information.

Arguments

- ***value_name***

  A required string that specifies register value variable. The variable holds a value to load into the register. The register value variable is used in the procedure file to load the value into a register through the data input pin.

- ***value_string***

  A required state string in a particular radix. The default is binary radix.

- -Radix {**Binary** | Decimal | Octal | Hexadecimal}

  An optional switch and literal that specifies a different radix.

Example

The following example deletes all register value variables, specifies a register value variable with a decimal radix and register width of 37, and then sets the value of the register.

delete_register_value -all

add_register_value setup_TDI1 0 -radix decimal -width 37

set_register_value setup_TDI1 0000000000000000000000000000000000111

report_register_value

```
//  Register value name                value string                width
//  -------------------  -------------------------------------  -----
//        setup_TDI1      0000000000000000000000000000000000111   37
```

Related Commands

add_register_value

delete_register_value

report_register_value

set_registered_io

Context: dft -scan

Mode: setup, analysis

Prerequisites: A scan model must be defined with the add_cell_models command before using this command.

Usage

set_registered_io {[-Exclude *pin_names*…] | [-INClude *pin_names*...]}
[-INPUT_Model *model_name*] [-OUTPUT_Model *model_name*]
[-INPUT_Clock *pin_pathname*] [-OUTPUT_Clock *pin_pathname*]
[-REG_Floating] [-REG_Comb_feedthrough] [-REG_Seq_feedthrough]
[-IGNORE_ISolation_cells [Input | Output]] [-IGNORE_Comb_path]

Description

Turns on the automatic registration of the primary I/O pins of the design.

The registered cells are also placed into scan chains.

Certain I/O pins are excluded from the automatic registration; the following is a comprehensive list of such exclusions, most of which are described in greater detail later.

- The system clock pins such as set, reset and clock, and test-related pins such as scan-ins/scan-outs, scan enable and test enable pins are excluded from the registration automatically.
- Pins listed with the -Exclude switch are excluded from the registration automatically.
- By default, I/Os that are driving/driven by scan-in/scan-out pins are not registered unless they are explicitly added to the -Include list.
- By default, unconnected I/Os are not registered.
- By default, combinational feed-throughs are not registered.
- By default, sequential feed-throughs are not registered.
- I/Os that have been already identified during the wrapper cells analysis with the shared wrapper cells are excluded from the registration.
- PIs driving the enable signals of the power isolation cells are excluded from the registration (note that this is applicable only for low-power designs where the power data is specified through a CPF or UPF file).
- PIs and POs in the combinational-logic-only paths are excluded from registration when the optional switch -Ignore_comb_path is specified (by default, these I/Os are registered with the dedicated wrapper cells, as described below in the description of this switch).

The specified primary I/O ports are registered with the dedicated wrapper cells. By default, the tool will insert a dedicated wrapper cell, which is a combination of a mux cell and a scan cell,

where one input of the mux is connected to the functional path of the I/O being registered and the other input is connected to the scan cell. This is done to ensure that the functional mode behavior is not changed and no clock latency is introduced on the functional path. The scan cell model to use for the dedicated wrapper cells is specified through the add_cell_models command.

To override the default behavior, use the set_registered_io command with the -Input_model and -Output_model switches and explicitly specify the Input and Output wrapper cell models. The tool will then insert instances of the specified models without the mux cells to register the I/Os.

By default, the dedicated wrapper cells are treated as core cells and are stitched into the appropriate core scan chains. To stitch the added dedicated wrapper cells into the separate Input and Output wrapper chains, the set_registered_io command should be used in conjunction with the analyze_wrapper_cells command. When the analyze_wrapper_cells command is issued, the dedicated wrapper cells added to the primary Inputs will be stitched into the appropriate Input wrapper chains, while the dedicated wrapper cells added to the primary Outputs will be stitched into the appropriate Output wrapper chains along with any identified shared wrapper cells.

By default, I/Os that are designated as scan-in/scan-out pins or are driving/driven by scan-in/scan-out pins are not registered unless they are explicitly added to the -Include list of this command. During I/O registration, the tool will report scan-in/scan-out related I/Os that are not explicitly listed with the -Include switch as not being registered. The -Include switch cannot be used in conjunction with the -Exclude switch.

A clock to control an added dedicated wrapper cell is chosen in the following order:

- A user-specified clock, if specified through the optional -Input_clock and -Output_clock switches.
- A downstream clock, which is a capture clock controlling the highest number of downstream flops of a primary I/O pin being registered.
- The most used clock of the corresponding wrapper type, which is a clock controlling the highest number of the Input or Output shared wrapper cells depending on the type of the dedicated wrapper cell.
- A test clock, which is either the user-specified test clock or a default test clock.

By default, unconnected I/Os are not registered. You can specify that unconnected I/Os should be registered by doing one of the following:

- Specify the -REG_floating switch for this command.
- Specify the unconnected I/Os in the -Include list of this command.

By default, combinational feed-throughs are not registered. A combinational feed-through is a path between a PI and PO containing only buffers and/or inverters or no logic at all. You can specify that combinational feed-throughs should be registered by doing one of the following:

- Specify the -REG_Comb_feedthrough switch for this command.

- Specify the PI/POs of the combinational feed-throughs in the -Include list of this command.

By default, sequential feed-throughs are not registered. A sequential feedthrough is a path between a PI and PO that has only one sequential cell and contains no combinational logic other than buffers and inverters. You can specify that sequential feed-throughs should be registered by doing one of the following:

- Specify the -REG_Seq_feedthrough switch for this command.
- Specify the PI/POs of the sequential feed-throughs in the -Include list of this command.

Note

This feature does not address the stitching of multiple cores, but rather the insertion of wrapper chains in a single core.

Arguments

- -Exclude *pin_names*

  An optional switch and repeatable string that specifies primary I/O pins to exclude from registration. Note that clock and scan-related pins are automatically excluded. This switch cannot be used with the -Include switch.

- -Include *pin_names*

  An optional switch and repeatable string that specifies which primary I/O pins to register. Clock and scan-related pins are automatically excluded from registration. This switch cannot be used with the -Exclude switch.

- -Input_model *model_name*

  An optional switch and string pair that uses model_name cells for registering primary inputs. You must first define model_name with the add_cell_models command.

- -Output_model *model_name*

  An optional switch and string pair that uses model_name cells for registering primary outputs You must first define model_name with the add_cell_models command.

- -Input_clock *pin_pathname*

  An optional switch and string pair that specifies to use pin_pathname clock to control registration cells inserted for input pins. The specified pin pathname should either be a primary input or a top-level instance output pin.

- -Output_clock *pin_pathname*

  An optional switch and string pair that specifies to use the pin_pathname clock to control registration cells inserted for output pins. The specified pin pathname should either be a primary input or a top-level instance output pin.

- -REG_Floating

  An optional switch that specifies to register unconnected I/Os. By default, unconnected I/Os are not registered.

- -REG_Comb_feedthrough

  An optional switch that specifies to register combinational feed-throughs. A combinational feed-through is a path between a PI and PO containing only buffers and/or inverters or no logic at all. By default, combinational feed-throughs are not registered. Note: A multi fan-out PI is considered a combinational feed-through only when all of its fan-outs are combinational feed-throughs.

- -REG_Seq_feedthrough

  An optional switch that specifies to register sequential feed-throughs. A sequential feedthrough is a path between PI and PO that has only one sequential cell and contains no combinational logic other than buffers and inverters. By default, sequential defaults are not registered. Note: A multi fan-out PI is considered a sequential feed-through only when all of its fan-outs are sequential feed-throughs.

- -IGNORE_ISolation_cells [Input | Output]

  An optional switch that specifies to insert the dedicated input wrapper cells immediately after the primary Input ports being identified and to insert the dedicated output wrapper cells immediately before the primary Output ports being identified. By default, the dedicated input wrapper cells are inserted after the power isolation cell (if any encountered during the primary input ports identification) and the dedicated output wrapper cells are inserted before the power isolation cells (if any encountered during the primary output ports identification).

  Note that this switch is valid only for low-power designs where the power data is specified through a CPF or UPF file.

- -IGNORE_Comb_path

  An optional switch that prevents addition of the dedicated wrapper cells to the PIs and POs in the combinational-logic-only paths. When the “set_registered_io -IGNORE_Comb_path” command is used together with the “set_wrapper_chains -IO_registration ON” command, the tool keeps on identifying all shared wrapper cells (if any) that are reachable from PIs and POs in the combinational-logic-only paths. A non-scan cell or a black box in the path does not terminate the shared wrapper cell identification for PIs and POs in the combinational-logic-only paths. By default, PIs and POs in the combinational-logic-only paths are registered with the dedicated wrapper cells.

Example 1

The following example illustrates how scan-based I/O registration can be used along with wrapper chains:

```
add_cell_model FDSQ -type scancell CLK D
set_wrapper_chains -exclude in1 in2 out1 out2 -input_number 1 -output_number 1
set_registered_io -include in1 in2 out1 out2
set_system_mode analysis
analyze_wrapper_cells
report_wrapper_cells
insert_test_logic
```

In this example, the tool identifies the input and output wrapper cells upon issuing the analyze_wrapper_cells command. Four I/O pins are excluded from identification by the set_wrapper_chains command and explicitly included for I/O registration by the set_registered_io command.

The input registration cells are placed into the input wrapper chains and the output registration cells are placed into the output wrapper chains. For more information, see the set_wrapper_chains command. Note that this registration allows you to insert registration cells along with the identified wrapper cells.

The scan cell used for registration can be modeled in many ways as long as it has the same input and output pins as a scan cell. Figure 7-36 shows a scan cell that is used as a control and observe point. When used as an observe point, it can capture data by means of the feedback connection. Also, note that the input registration cell uses the input wrapper chain scan enable signal, whereas the output registration cell uses the output wrapper chain scan enable signal.

Figure 7-36. Scan Cell Used as a Control and Observe Point

Example 2

In this example, the following scheme is used to identify different types of objects:

- Core flip-flop:

- Power isolation cells are indicated as:
- Dedicated wrapper cell:
- Shared wrapper cell:

Given a design with the following three power domains:

- P0 — always-ON domain (AON)
- P1 — ON/OFF domain (ONO)
- P2 — less-ON domain

The power isolation cells might be added as shown in Figure 7-37:

Figure 7-37. Power Isolation Cells in Design Block with Three Power Domains

Given the power isolation cells in Figure 7-37, the tool inserts dedicated wrapper cells, if needed, as shown in Figure 7-38.

Figure 7-38. Dedicated Wrapper Cell Insertion When Power Isolation Cells Present

As shown in Figure 7-38, the dedicated wrapper cells on the power domain output side are inserted internally to the power isolation cells; no changes are made to the insertion of dedicated wrapper cells on the power domains input side.

Related Commands

add_cell_models
analyze_wrapper_cells
set_name_prefix
set_wrapper_chains

set_relevant_coverage

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

Mode: setup, analysis

Usage

set_relevant_coverage [{-Include | -Exclude} { *fault_class*[.*fault_subclass*] …}]

Description

Determines which sub-class(es) of untestable faults are included in or excluded from the relevant coverage reported by the tool.

Issue this command without any arguments to report the current settings.

Arguments

- -Include

  An optional switch that includes the specified fault sub-class(es) in the relevant coverage reported by the tool.

- -Exclude

  An optional switch that excludes the specified fault sub-class(es) from the relevant coverage reported by the tool.

- *fault_class*[.*fault_subclass*]

  An optional, repeatable string that specifies the fault class or fault class/sub-class to include or exclude. Table 7-49 lists the fault classes that you can specify with this command and whether they are included in the reported relevant coverage by default. Table 7-50 lists the available fault sub-classes. You can specify a fault class or sub-class either by name or code. For AU faults, you can define a customized sub-class using a unique name and instruct the tool to include (default) or exclude this user-defined sub-class from the relevant fault coverage. For information about using a user-defined sub-calls, refer to “Example 2” on page 1922.

Table 7-49. Fault Classes for Relevant Coverage

Name	Code	Included in Relevant Coverage by Default?
ATPG_UNTESTABLE	AU	yes
REDUNDANT	RE	no
TIED	TI	no
BLOCKED	BL	no
UNUSED	UU	no

Table 7-50. Fault Sub-Classes for Relevant Coverage

Fault Class	Fault Sub-Class	Code	Included in Relevant Coverage by Default?
AU	BLACK_BOXES	BB	yes
	CELL_CONSTRAINTS	CC	yes
	EDT_BLOCKS	EDT	no
	FALSE_PATHS	FP	no
	MULTICYCLE_PATHS	MCP	yes
	PIN_CONSTRAINTS	PC	yes
	SEQUENTIAL_DEPTH	SEQ	yes
	TIED_CELLS	TC	yes
	USER-DEFINED	none	yes
UU	CONNECTED[1]	CON	no
	FLOATING[1]	FL	no

1. You cannot report or display details for this sub-class unless you first enable the sub-class analysis using the set_fault_subclass_analysis command.

Example 1

The following example removes the PIN_CONSTRAINTS and CELL_CONSTRAINTS sub-classes from relevant coverage reported by the tool:

```
set_relevant_coverage -exclude AU.PC AU.CC
set_relevant_coverage
```

```
Relevant Coverage Setting Report
----------------------------------------
    Fault Sub-class      Relevant Coverage
-----------------------  -----------------
AU.BB                           yes
AU.EDT                           no
AU.PC                            no
AU.TC                           yes
AU.CC                            no
AU.FP                            no
AU.MCP                          yes
UU.FL                            no
UU.CON                           no
TI                               no
BL                               no
RE                               no
----------------------------------------
```

Example 2

The following example shows how to use a user-defined fault sub-class to exclude specific pins from reporting. This example is based on the following fault list contained in *flist.au.usr1*:

```
//    type    code    pin_pathname
//    ----   ------   ------------------------------------
      0     AU.USR1      /si7
      0     EQ.USR1      /sf_1_7/SI
      1     AU.USR1      /si7
      1     EQ.USR1      /sf_1_7/SI
      0     AU.USR1      /si8
      0     EQ.USR1      /sf_1_8/SI
      1     AU.USR1      /si8
      1     EQ.USR1      /sf_1_8/SI
```

This fault list creates a user-defined sub-class for AU faults on two pins for both SA0 and SA1.

read_faults flist.au.usr1 -retain
set_relevant_coverage -exclude AU.USR1
report_statistics

```
                              Statistics Report
                               Stuck-at Faults
---------------------------------------------------------------------------
Fault Classes                                      #faults     #faults
                                                   (total)  (total relevant)
------------------------------------------ ------------- ---------------
  FU (full)                                          1397        1389
  ------------------------------------------ ------------- ---------------
  UC (uncontrolled)                                   27 ( 1.93%)   same ( 1.94%)
  DS (det_simulation)                                728 (52.11%)   same (52.41%)
  DI (det_implication)                               315 (22.55%)   same (22.68%)
  UU (unused)                                        234 (16.75%)   same (16.85%)
  AU (atpg_untestable)                                93 ( 6.66%)     85 ( 6.12%)
---------------------------------------------------------------------------
```

```
Untested Faults
  -------------------------------------------------------
  AU (atpg_untestable)
    BB (black_boxes)                          12 ( 0.86%)   same ( 0.86%)
    PC* (pin_constraints)                     20 ( 1.43%)   same ( 1.44%)
    TC* (tied_cells)                          53 ( 3.79%)   same ( 3.82%)
    USR1                                       8 ( 0.57%)     deleted
*Use "report_statistics -detailed_analysis" for details.
---------------------------------------------------------------------------
Coverage
  -------------------------------------------------------
  test_coverage                                 89.68%            90.30%
  fault_coverage                                74.66%            75.09%
  atpg_effectiveness                            98.07%            98.06%
```

Related Commands

report_statistics

set_scan_enable

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

set_scan_enable [*scan_enable_pin_pathname* [-isolate]] [*primary_input*]
[-Active {High | Low}]
[-Chain *chain_name*... | -Wrapper_chain [*chain_name*... | -INput | -OUTput] | -Partition *partition_name*...] [-Clock *clock_pin*]

Description

Assigns scan_enable signals to specific scan chains.

Scan chains can be specified by either name, wrapper chain type, scan partition, or clock domain. If no scan chains are specified, the specified scan_enable signal is assigned to all scan chains.

The set_scan_enable command uses either the default scan enable signal names or allows you to assign a new scan enable signal name to all or to just the specified scan chains using the scan_enable_pin_pathname argument. Table 7-51 shows the default names for three types of scan enable signals used in the tool.

Table 7-51. Default Scan Enable Signal Names

Default Name	Description
scan_en	Scan enable for mux-DFF type (core scan cells)
scan_en_in	Scan enable for mux-DFF type (input wrapper cells)
scan_en_out	Scan enable for mux-DFF type (output wrapper cells)

The tool creates three types of scan enable signals: one for core scan cells, one for input wrapper cells, and one for output wrapper cells. You can override the default base names for each scan enable signal type using one of the following commands:

set_scan_enable <scan_enable_pin_pathname>

set_scan_enable <scan_enable_pin_pathname> -Wrapper_chain -Input

set_scan_enable <scan_enable_pin_pathname> -Wrapper_chain -Output

The set_scan_enable command can be issued in a sequence to either refine the scan enable signals assignments or overwrite the previous assignments (see the example section for this command). In general, the following rules are applied to determine how signal assignments are affected by subsequent commands:

- The most recent command that assigns a scan_enable signal takes precedence.
- If the most recent command operates on a disjoint set of scan chains, then the previous scan enable signal assignments remain intact.

- If the most recent command operates on previously specified scan chains, then previous scan_enable signal assignments are overwritten.

Arguments

- *scan_enable_pin_pathname* [-Isolate]

  An optional string that specifies a pin pathname for the scan_enable signal driver. The specified pin can be either a top-level scan enable port or an internal instance pin (connection node). If an internal instance pin is specified, it must trace back to a primary input via a simple path (only inverters or buffers) or the primary_input argument.

  -Isolate — Isolates new fanouts of the specified scan enable signal. Each new fanout connection is gated by an AND or NOR gate and controlled by the global test enable signal. This switch is applicable to a scan enable signal driven by a top-level port or a top-level internal instance pin only.

  If no scan_enable signal driver is specified, the default scan enable name is used: scan_en, scan_en_in, scan_en_out.

- *primary_input*

  An optional string that specifies a top-level scan_enable port. This argument supplies a primary input/top-level port for an internal instance pin specified by the scan_enable_pin_pathname argument. The specified top-level port is used when generating the ATPG dofile and test procedure files. If the specified top-level port does not exist, it is created. The specified top-level port must be a primary input port.

- -Active {High | Low}

  An optional switch and literal pair that specifies whether the scan_enable signal is active low or high. The default setting is high.

- -CHain *chain_name*…

  An optional switch and a repeatable string that identifies individual scan chains to assign the specified scan_enable signal to.

- -Wrapper_chain [*chain_name*… | -INPut | -OUTput]

  An optional switch and a repeatable string or literal pair that identifies the wrapper chains to assign a specified scan_enable signal to. Options include:

  chain_name... — Specifies one or more wrapper chain names.

  -INPut — Specifies all input wrapper chains when two-domain distribution is used.

  -OUTput — Specifies all output wrapper chains when two-domain distribution is used.

  The -Input | -Output options should be used if the specified scan enable signal will be associated with either all input wrapper chains or all output wrapper chains, respectively.

  When the -Wrapper_chain switch is issued without arguments, the specified scan enable signal is assigned to all wrapper chains when one-domain distribution is used.

  Wrapper chain creation must be enabled. For more information on distribution modes and creating wrapper chains, see the add_input_constraints command.

This switch, along with either the -Input or -Output option, can be used in conjunction with the -Clock switch. In this case, the specified scan enable signal is assigned only to the wrapper chains that belong to both the specified type of wrapper chains and the specified clock domain.

- -Partition *partition_name*…

  An optional switch and repeatable string pair that specifies names of scan partitions added using the add_scan_partition command. Use this option to assign a specified scan_enable signal to the scan chains created within one or more partitions.

  This option can be used in conjunction with the -Clock switch. In this case, the specified scan enable signal is assigned only to the scan chains that belong to both the specified scan partition and the specified clock domain.

- -Clock *clock_pin*

  An optional switch and string pair that associates the specified scan enable signal with the specified clock (clock domain). Clock_pin can be either an existing top level port (primary input pin) or an existing internal pin pathname.

  This switch can be used in conjunction with either the -Wrapper_chain or -Partition option, in which case a unique scan enable signal is generated just for the scan chains that belong to both the specified wrapper chain type or partition and the specified clock domain. This switch is ignored when it is used in conjunction with the -Chain option.

  An error message is issued when this switch is specified with either the -Clock Merge or the filename -Fixed option of the insert_test_logic command.

  This switch can be used in conjunction with the -Edge Merge option of the insert_test_logic command.

Example 1

Assuming two-domain distribution of the identified wrapper cells, the following example uses a sequence of set_scan_enable commands to make all input wrapper chains controllable via the insen1 signal, and all output wrapper chains controllable via the outsen1 signal. All of the commands operate on disjoint sets of scan chains; therefore, each command affects different scan chains and does not override scan_enable assignments made by the previous command.

```
set_scan_enable insen1 -wrapper_chain -input
set_scan_enable outsen1 -wrapper_chain -output
```

Example 2

The following example defines two scan partitions: partA and partB. A single scan chain is inserted by default for partA and two scan chains are inserted for partB. Two scan chains are inserted for the remaining cells in the default scan partition, as specified by the -number argument of the insert_test_logic command.

```
set_scan_enable sen
set_scan_enable senPartA -partition partA
set_scan_enable senPartB -partition partB

set_system_mode analysis
add_scan_partition partA -instance udff1 umodA/udff2 umodB/udff3 // 1 chain
add_scan_partition partB -instance umodC -number 2 // 2 chains

insert_test_logic -number 2 // 2 chains inserted for the default partition
```

The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command refines the first command and makes scan chains of partA controllable via the senPartA scan enable signal. The third set_scan_enable command further refines the first command and makes scan chains of partB controllable via the senPartB scan enable signal.

The second command operates on a set of scan chains that is entirely within the set specified in the first command; therefore, the scan chains that are in both sets will get the most recent assignment. The third command operates on a set of scan chains that is disjoint from the set in the second command but is entirely contained within the first set; therefore, the scan chains that are in both the first and the third sets get the most recent assignment.

Example 3

The following example defines two scan partitions: partA and partB. The first set_scan_enable command makes all scan chains controllable via the sen scan enable signal. The second set_scan_enable command specifies an internal pin, /modX/i_sen/x, associated with the primary input, senPartA, as a driver of the scan enable signal controlling all scan chains of partA. The third set_scan_enable command specifies an internal pin, /modY/i_sen/x, associated with the primary input, senPartB, as driver of the scan enable signal controlling all scan chains of partB.

```
add_clocks 0 /clkInput

set_scan_enable sen
set_scan_enable /modX/i_sen/x senPartA -partition partA
set_scan_enable /modY/i_sen/x senPartB -partition partB
set_scan_enable senClk -clock clkInput

set_system_mode analysis
add_scan_partition partA -instance udff1 umodA/udff2 umodB/udff3 // 1 chain
add_scan_partition partB -instance umodC -number 2 // 2 chains
insert_test_logic -number 2 // 2 chains inserted for the default partition
```

The last set_scan_enable command specifies that all scan chains in the clkInput clock domain are controllable by the senClk scan enable signal. The second and the third set_scan_enable commands overwrite the assignments of the first set_scan_enable command affecting chains that are in partA and partB. Since the second and the third set_scan_enable commands operate on disjoint sets, they do not affect previous assignments.

The fourth command will overwrite some of the assignments of the second and the third commands for scan chains that are in partA and partB and also in the *clkInput* clock domain. If it is desirable to restrict the clock domain's assignments to a specific partition, the -Partition and

-Clock options should be issued in conjunction in the same set_scan_enable call as shown in Example 4.

Example 4

In this example, two scan partitions are defined: partA and partB. This example uses a set_scan_enable_sharing command to make all scan chains within each scan partition controllable via a unique scan enable signal partSenN where N is a unique number for each partition. The set_scan_enable command specifies to assign a unique scan enable signal, clkSen, to only the scan chains inside the partB scan partition that are also in the clock1 clock domain. The second call operates on a set of scan chains that is the same as the set of scan chains in the first call; therefore, certain previous assignments that are subject to the most recent assignment are overwritten.

```
add_clocks 0 clk1
set_scan_enable_sharing -Prefix partSen -Scan_partition

set_scan_enable clkSen -Partiton partB -Clock clock1

set_system_mode analysis
add_scan_partition partA -instance udff1 umodA/udff2 umodB/udff3 // 1 chain
add_scan_partition partB -instance umodC -number 2 // 2 chains
```

Related Commands

add_scan_partition
add_input_constraints
report_scan_enable
set_scan_insertion
set_scan_enable_sharing

set_scan_enable_sharing

Context: dft -scan

Mode: setup, analysis

Prerequisites: mux-DFF scan type

Usage

set_scan_enable_sharing [-Prefix *base_name*] [-Active {High | Low}]
{[-Max_number_of_chains *integer*]
[-Input_wrapper_chain | -Output_wrapper_chain | -Scan_partition] [-Clock_domain]}

Description

Divides all scan chains into specified groups and assigns a unique scan_ enable signal to each group.

Scan chains can be grouped by:

- Specifying a maximum number of chains in each group, by wrapper chain types, by scan partitions, or by clock domains.
- Specifying a maximum number of chains within groups created by wrapper chain types, by scan partitions, or by clock domains.
- Clock domains within groups created by wrapper chain types or by scan partitions.

The set_scan_enable_sharing command can be issued in a sequence to either refine the scan enable signals assignments or overwrite the previous assignments (see the example section for this command). In general, the following rules are applied to determine how signal assignments are affected by subsequent commands:

- The most recent command that assigns a scan_enable signal takes precedence.
- If the most recent command operates on a disjoint set of scan chains, then the previous scan enable signal assignments remain intact.
- If the most recent command operates on previously specified scan chains, then previous scan_enable signal assignments are overwritten.

Arguments

- -Prefix *base_name*

  An optional switch and string pair that specifies a base name (prefix) used in combination with sequential numbers to automatically generate unique top-level scan_enable signals (*base_name*N) for specified groups of scan chains.

  If this option is not specified, an appropriate default scan enable name (scan_en, scan_en_in, scan_en_out) is used as the base name (prefix).

- -Active {High | Low}

  An optional switch and literal pair that specifies whether the scan_enable signal is active low or high.

- -Max_number_of_chains *integer*

  A required switch and integer or literal pair that divides scan chains into groups. Options include:

  integer — Groups chains by a specified integer, where each group cannot have more than the integer number of scan chains. A unique scan_enable signal is then generated and assigned to each group.

- -Input_wrapper_chain | -Output_wrapper_chain

  Optional switches specifying to generate a unique scan enable signal for either all input wrapper chains or all output wrapper chains. When the switches are used in conjunction with the -Max_number_of_chains option, the number specified by integer is applied only to the specified type of wrapper chains. The generated scan enable signals are treated as SEN_IN or SEN_OUT type, respectively.

  -Input_wrapper_chain — Used to specify grouping for input wrapper chains. When this switch is used in conjunction with the Max_number_of_chains switch, the number specified by integer is applied to only input wrapper chains. Only valid when input wrapper chains are defined. For more information, see the add_input_constraints command.

  -Output_wrapper_chain — Used to specify grouping for output wrapper chains. When this switch is used in conjunction with the -Max_number_of_chains switch, the number specified by integer is applied to only output wrapper chains. Only valid when output wrapper chains are defined. For more information, see the add_input_constraints command.

  These switches can be used in conjunction with the -Clock_domain switch.

- ***-Scan_partition***

  An optional switch that specifies groupings for each scan partition added using the add_scan_partition command.

  With this switch, a unique scan enable signal is generated for every scan partition (that is, for scan chains within each partition). When this switch is used in conjunction with the -Max_number_of_chains switch, the number specified by integer is applied to each scan partition.

  This switch can be used in conjunction with the -Clock_domain switch.

- -Clock_domain

  An optional switch that specifies to generate a unique scan enable signal for each clock domain. When this switch is used in conjunction with the -Max_number_of_chains switch, the number specified by integer is applied to each clock domain.

This switch can be used in conjunction with either the -Input_wrapper_chain or -Output_wrapper_chain option; in this case a unique scan enable signal is generated only for each clock domain within the input or output wrapper chains.

This switch cannot be used in conjunction with the -Clock Merge or the filename -Fixed switches of the insert_test_logic command; in this case an error message is issued.

This switch can be used in conjunction with the -Edge Merge switch of the insert_test_logic command.

Example 1

The following example uses a sequence of set_scan_enable_sharing commands to make every group of 3 input wrapper chains controllable via a unique scan_enable signal, insenN, and every group of 5 output wrapper chains controllable via a unique scan enable_signal, outsenN.

set_scan_enable_sharing -Prefix insen -Max_number_of_chains 3 -Input_wrapper_chain
set_scan_enable_sharing -Prefix outsen -Max_number_of_chains 5 -Output_wrapper_chain

The second command operates on a set of scan chains that is disjoint from the set of scan chains operated on by in the first command, so the previous assignments remain intact.

Example 2

The following example defines 3 scan partitions: spar1, spar2, spar3. Two scan chains are created for spar1, 6 scan chains for spar2, and 3 scan chains for spar3. Two scan chains are created in the default scan partition for the remaining cells, as specified by the -number argument of the insert_test_logic command.

set_scan_enable_sharing -prefix SENPAR -scan_partition
set system mode dft
add_scan_partition spar1 -ins a2/e* -number 2 -verbose // 2 chains: chain1, chain2
add_scan_partition spar2 -ins a1/b2 a1/b1/e2 -number 6 -verbose // 6 chains: chain3, chain4, chain5, chain6, chain7, chain8
add_scan_partition spar3 -mod C -number 3 -verbose // 3 chains: chain9, chain10, chain11
insert_test_logic -number 13 // 2 chains for the default partition: chain12, chain13
report_scan_enable

```
-----------------------------------------------------------------------
--
//  command: report_scan_enable
-----------------------------------------------------------------------
--
Primary Input  Internal Connection Node  Scan Chain
-----------------------------------------------------------------------
--
/SENPAR1       --                   chain12
                                    chain13
-----------------------------------------------------------------------
--
/SENPAR2     --                     chain1
                                    chain2
-----------------------------------------------------------------------
--
/SENPAR3     --                     chain3
                                    chain4
```

```
                                    chain5
                                    chain6
                                    chain7
                                    chain8
-----------------------------------------------------------------------
--
/SENPAR4      --                    chain9
                                    chain10
                                    chain11
-----------------------------------------------------------------------
--
```

A unique scan_enable signal, SENPARN, is generated and assigned to all core scan chains.

Example 3

The following example uses a sequence of set_scan_enable_sharing commands to make every group of three input wrapper chains controllable via a unique scan enable signal insenN, where N is a unique number for each group, and every group of five output wrapper chains controllable via a unique scan enable signal outsenN, where N is a unique number of each group.

set_scan_enable_sharing -Prefix insen -Max_number_of_chains 3 -Input_wrapper_chain

set_scan_enable_sharing -Prefix outsen -Max_number_of_chains 5 -Output_wrapper_chain

The second command operates on a set of scan chains that is disjoint from the set of scan chains in the first command; therefore, the previous assignments are not affected by the subsequent assignments.

Example 4

In this example, two scan partitions are defined: partA and partB. This example uses a set_scan_enable_sharing command to make all scan chains within each scan partition controllable via a unique scan enable signal partSenN, where N is a unique number for each partition. Next, the set_scan_enable command specifies to assign a unique scan enable signal, clkSen, to only the scan chains inside the partB scan partition that are also in the clock1 clock domain. The set_scan_enable command operates on a set of scan chains that is the same as the set of scan chains in the set_scan_enable_sharing command; therefore, certain previous assignments that are subject to the most recent assignment are overwritten.

add_clocks 0 clk1
set_scan_enable_sharing -Prefix partSen -Scan_partition

set_scan_enable clkSen -Partition partB -Clock clock1

set_system_mode analysis

add_scan_partition partA -instance udff1 umodA/udff2 umodB/udff3 // 1 chain

add_scan_partition partB -instance umodC -number 2 // 2 chains

Related Commands

add_scan_partition
report_scan_enable
set_scan_enable
add_input_constraints
set_scan_insertion

set_scan_insertion

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_scan_insertion [-TEn *pathname*] [-TClk *pathname*] [-SClk *pathname*]
[-SMclk *pathname*] [-SSclk *pathname*] [-SET *pathname*] [-RESet *pathname*]
[-Write *pathname*] [-REAd *pathname*] [-WEn *pathname*]
[-Muxed | -Disabled] [-Active {High | Low}]

Description

Specifies the input control signals' drivers used by the insert_test_logic command.

Additional control pins are specified with set_scan_enable, set_bidi_gating, and set_tristate_gating commands.

The tool uses the write control and read control pins when inserting test logic for RAMs. DRC verifies whether the RAMs can be held off when the write control pin is off. No test logic is inserted in RAMs that can be held off.

If the new scan design is intended for Tessent FastScan/Tessent TestKompress, and the write clock input of the RAM is not controllable, it is multiplexed out by a new write clock. The selector to the mux is the test enable pin.

You can use instance pin pathnames to define the test enable and clock pin names. The instance pathnames must trace back through a simple path (inverters or buffers only) to a primary input/output of the design, or you must manually develop the test procedure file and the dofile before running ATPG.

Arguments

- -TEn *pathname*

  An optional switch and string pair that specifies a name for the test enable pin. By default, the test enable pin is named test_en.

  Use the -Active switch to specify the active state of the Test enable signal.

- -TClk *pathname*

  An optional switch and string pair that specifies a pathname for the test clock pin for all scan types. If no test clock pin exists, one is created using the specified name. By default, the new test clock pin is named test_clk.

  If no test clock pin is specified, an existing clock pin from the design is used. Eligible clock pins must be defined with the add_clocks command and capable of capturing data in a sequential cell without test logic in the clock path. The clock pin is selected at random. If no clock pins are found, a new one is created and named test_clk.

If an internal pin is specified as the test clock pin, it is traced back to a predefined scan clock. If a predefined scan clock is found, the off-state value is used for test logic insertion. This tracing can only be performed along a single path that consists of only straight wire, inverters, and buffers.

- -SClk *pathname*

  An optional switch and string pair that specifies a pathname for the clock pin in clock-scan type scan. A scan clock pin is only created for the scan clock port in the dual-port type mux scan. By default, the new scan clock pin is named scan_clk.

- -SMclk *pathname*

  An optional switch and string pair that specifies a pathname for the scan master clock pin in LSSD type scan. By default, the scan master clock pin is named scan_mclk.

- -SSclk *pathname*

  An optional switch and string pair that specifies a pathname for the scan slave clock pin in LSSD type scan. By default, the scan slave clock pin is named scan_sclk.

- -SET *pathname*

  An optional switch and string pair that specifies a pathname for the set pin used for the flip-flop or latch when inserting test logic. By default, the set pin is named scan_set.

  Use the -Active switch to specify the active state of the Set signal.

- -RESet *pathname*

  An optional switch and string pair that specifies a pathname for the reset pin used for the flip-flop or latch when inserting test logic. By default, the set pin is named scan_reset.

  Use the -Active switch to specify the active state of the Reset signal.

- -Write *pathname*

  An optional switch and string pair that specifies a pathname for the write control pin used for RAM when inserting test logic. By default, the write control pin is named write_clk.

- -REAd *pathname*

  An optional switch and string pair that specifies a pathname for the read control pin used for RAM when inserting test logic. By default, the read control pin is named read_clk.

- -WEn *pathname*

  An optional switch and string pair that specifies a pathname for the extest mode pin that drives the enable pin of the multiplexers inserted at the scan input pins of wrapper chains. By default, the extest mode pin is named “wrapper_extest”. For more information on inserting extest ports for wrapper chains, please see the set_wrapper_chains command.

- -Muxed | -Disabled
  An optional switch that determines how the specified set, reset, write control, or read control

lines are gated. These options can only be used with -Set, -Reset, -Write, and -Read switches specified in the same command.

Options include:

-Muxed — Multiplexes all set and reset pins with the original signal. Only set and reset pins defined as clocks are affected. Default setting.

-Disabled — Uses an AND gate with the test enable signal to disable the set and reset inputs of flip-flops and the SEN type scan enable signal to disable the write and read clocks.

- -Active High | Low

  An optional switch and literal pair that specifies the active state of the Test enable, Set or Reset signals. Within a single command, you can only use the -Active switch once.

  The -Active switch applies to any of the above signals that are specified in the command with the -Ten, -Set, and -Reset switches. By default, all these signals are active on high. The -Active switch is not applicable to any of the other signals specified in this command.

Example 1

The following example specifies an internal pin, **modxyz/nand02/test_en_L**, as the driver of the Test enable signal to be used during the scan insertion.

```
add_clocks 0 clock
set_system_mode analysis
set_scan_insertion -ten modxyz/nand02/test_en_L -active low
insert_test_logic -number 8
```

Example 2

The following example shows how to set different controls with successive set_scan_insertion commands.

```
set_scan_insertion -ten TEST_MODE
set_scan_insertion -reset RESET_L -active low
set_scan_insertion -write ATPG_WRT_INH -read ATPG_READ_INH
set_scan_insertion -set ATPG_SET -disabled
```

Related Commands

insert_test_logic
report_scan_pins
set_bidi_gating
set_scan_enable
set_scan_enable_sharing
set_tristate_gating
set_wrapper_chains

set_scan_pins

Context: dft -scan

Mode: setup, analysis

Usage

set_scan_pins {**Input** | **Output**} [-INDexed | -Bused] [-Prefix *base_name*] [-INItial *index#*] [-Modifier *incr_index#*] [-Suffix *suffix_name*] [-Range {Ascending | Descending}] [-All | -Core | -WRAPPER_Intest | -WRAPPER_Extest]

Description

Changes the scan-in or scan-out pin naming parameters to index or bus format.

The **set_scan_pins** command specifies the index or bus naming conventions for scan-in or scan-out pins. The report_environment command displays the names and values that the tool uses for scan-in and scan-out pins when inserting scan chains. Once you have inserted scan chains, you can use the report_scan_chains command to display the chosen index or bus naming convention for the scan-in pins and/or scan-out pins.

Indexed names are in the form:

```
base_name + index# + suffix_name
```

Bused names are in the form:

```
base_name + [index#]
```

Arguments

- **Input**

  A literal specifying that the tool apply the index or bus format on the scan-in pins.

- **Output**

  A literal specifying that the tool apply the index or bus format on the scan-out pins.

- -INDexed

  An optional switch specifying that the tool apply the index format to the scan-in or scan-out pin names. This is the default.

- -Bused

  An optional switch specifying that the tool apply the bus format to the scan-in or scan-out pin names.

- -Prefix *base_name*

  An optional switch and string pair that specifies the root name of the scan-in or scan-out pin. The default name is scan_in.

- -INItial *index#*

  An optional switch and integer pair that specifies the initial index value of the scan-in or scan-out pin name. The default value is 1.

- -Modifier *incr_index#*

  An optional switch and integer pair that specifies the incremental value that to add to the index# when creating additional names with the same base_name. The default is 1.

- -Suffix *suffix_name*

  An optional switch and string pair that specifies the name that you want to place after the index#. The tool uses this only for indexed naming. The default is null.

- -Range {Ascending | Descending}

  An optional switch and label pair that specifies whether the tool lists the range format of bused-type scan I/O port declarations in ascending or descending order. This switch is ignored if the –Indexed switch is specified.

  Ascending

  The bit range is listed as [m:n], where n is a larger integer than m. This is the default if -Range is not specified. A 4-bit scan input pin declaration with an ascending bit range format is written as “input [1:4] scan_in”.

  Descending

  The bit range is listed as [n:m], where n is a larger integer than m. A 4-bit scan input pin declaration with a descending bit range format is written as “input [4:1] scan_in”.

- –All

  An optional switch specifying that the format parameters specified by this command are to be applied to the scan-in or scan-out pins of all scan chains. This is the default behavior if the All, Core, Wrapper_intest, or Wrapper_extest switches are not specified.

- –Core

  An optional switch specifying that the format parameters specified by this command are to be applied to the scan-in or scan-out pins of the core type scan chains.

- –Wrapper_intest

  An optional switch specifying that the format parameters specified by this command are to be applied to the primary (regular or intest) scan-in or scan-out pins of the wrapper type scan chains.

- –Wrapper_extest

  An optional switch specifying that the format parameters specified by this command are to be applied to the auxiliary (extest) scan-in or scan-out pins of the wrapper type scan chains. The wrapper chain extest ports are those that are inserted when the set_wrapper_chains command is specified with the –Wrapper_extest_ports switch.

Examples

The following example configures scan insertion to use bus names for the scan-in pins and scan-out pins, with the index number starting at 5 and incrementing by 2 for scan-in pins, and the index number starting at 4 and incrementing by 2 for scan-out pins:

```
add_clocks 0 clock
set_system_mode analysis
set_scan_pins input -bused -prefix scin -initial 5 -modifier 2
set_scan_pins output -bused -prefix scout -initial 4 -modifier 2
insert_test_logic -number 7
```

Related Commands

insert_test_logic

set_screen_display

Context: unspecified, all contexts

Mode: all modes

Usage

set_screen_display **ON** | **OFf**

Description

Specifies whether the tool writes the transcript to the session window.

If you create a logfile with the set_logfile_handling command, you may want to disable the tool from writing the same information to the session transcript window.

Arguments

- **ON**

  A literal that enables the tool to write the session information to the transcript window. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that disables the tool from writing any of the session information to the transcript window, including error messages.

Example

The following example shows how to use the logfile functionality to capture the transcript in a file and then disable the tool from writing the transcript to the display.

```
set_logfile_handling /user/design/setup_file
set_screen_display off
```

Related Commands

set_logfile_handling

set_shadow_check

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

Mode: setup, analysis (dft -scan context only)

Usage

set_shadow_check **ON** | **OFf** | **REstricted**

Description

Specifies whether the tool will identify sequential elements as a “shadow” element during scan chain tracing.

You can use the set_shadow_check command to disable the checking and avoid corresponding error messages. This will prevent identification of any non-scan sequential element as a shadow element.

Arguments

- **ON**

  A literal that enables shadow checking. This is the initial state upon invocation of the tool.

- **OFf**

  A literal that disables shadow checking.

- **REstricted**

  A literal that disables the tool’s identification of shadow cells that require independent shifts in certain pattern formats (for example, parallel Verilog format).

Note

An independent shift results from a one-shift apply shift statement in the load_unload procedure. For more information, refer to the description of the P19 rule.

Example

The following example disables shadow checking.

```
set_shadow_check off
```

Related Commands

set_drc_handling
report_environment
write_patterns

set_shift_register_identification

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_shift_register_identification ON | OFF [-IGNORE_CONSTraint_values {ON|OFF}]

Description

Enables/disables shift register identification. The default is ON.

For more information about automatic identification of shift registers, see "Automatic Recognition of Existing Shift Registers" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **ON**

  A required literal that enables shift registration identification. This is the default.

- **OFf**

  A required literal that disables shift registration identification.

- **IGNORE_CONSTraint_values ON|OFF**

  An optional switch and literal pair that enables or disables ignoring the DRC calculated state-stability values during shift register identification. When the switch is On, the constraint values that sensitize a data path between sequential cells are ignored. As a result, multiple-input gates (AND, OR, MUX, etc.) between sequential cells cannot be traced during shift register identification. Single-input gates (BUF and INV), on the other hand, can always be traced. When the switch is Off, the constraint values that sensitize a data path between sequential cells are considered during shift register identification tracing. Multiple-input gates do not block the tracing as long as a single path is sensitized through them.

  OFF

  A literal that specifies considering the constraint values. This is the default.

  ON

  A literal that specifies ignoring the constraint values.

Example

The following example disables automatic shift register identification.

```
set_shift_register_identification off
```

Related Topics

report_scan_cells

report_shift_registers

set_simulation_options

Context: all contexts

Mode: setup, analysis

Usage

set_simulation_options [-SET_Reset_dominate_port { OFf | ON }]
[-C6_mask_races { OFf | ON }] [-MUX_Select_x_sim_x { OFf | ON }]
[-FEEDback_buffer_x_init_value { ON | OFf }]
[-REPORT_X_capturing_cells { OFf | ON }]

Description

Controls parallel simulation behavior.

The set_simulation_options command is a general command that controls the behavior of parallel simulation.

Arguments

- -SET_Reset_dominate_port OFf | ON

  An optional switch and literal pair that enables or disables setting the Set/Reset port as the dominate synchronous port for both flip-flops and latches when multiple ports are on at the same time. When the switch is On, and either Set or Reset port is on (but not both on), the evaluation value of the state element is dependent on the Set or Reset port. When the switch is Off and multiple ports are on at the same time, the evaluation value is dependent on all enabled ports. When enabled ports agree on the same value, the state element will be set to the agreed value. Otherwise, the state element is set to X. Also, the state element is set to X when Set and Reset port are both on simultaneously despite this switch setting.

  OFf

  A literal that specifies disabling set/reset as the dominate port.

  ON

  A literal that specifies enabling set/reset as the dominate port. This is the default.

- -C6_mask_races OFf | ON

  An optional switch to turn on pessimistic simulation on C6 violated state element and mask the capture value only when the data port transition at the same frame the clock port capture. The option allows you to change the handling of C6 violated cells when data and clock port have a race transition. By default, create_patterns will turn on clock_off simulation and the tool assumes that the clock port transition is earlier than the data port. If this assumption is not true, this option is recommended to turn on to mask the C6 violated cell to prevent mismatch due to race. This functionality is less pessimistic than command add_cell_constraints -DRC C6 as the later command will mask C6 violated cells even no races happen.

  OFf

A literal that disables this functionality. This is the default.

ON

A literal that enables this functionality.

- -MUX_Select_x_sim_x OFf | ON

  An optional switch and literal pair that specifies to model _mux primitives in the design as either consensus or non-consensus. By default, the tool always models _mux as consensus (evaluates _mux with a known value as long as both data ports are the same known value).

  You can use the ON argument to model the _mux primitive as non-consensus (output X for every _mux instance whose select line is X). Note that changing _mux primitives to non-consensus can reduce the test coverage. Also, you can use this switch in any mode, but when you use in setup mode, the tool evaluates _mux instances as non-consensus mux instances, which can result in additional DRC violations.

- -FEEDback_buffer_x_init_value ON | OFf

  An optional switch and literal pair that allows the FB_BUF primitive to carry over its value from the previous cycle instead of being initialized to X. The default is ON, which means that the FB_BUF is still initialized with X value at the beginning of a simulation cycle. You can choose to turn off X initialization in order to get less pessimistic simulation results. Note that turning off this option will increase the chance of pattern simulation mismatches if all FB_BUF primitives in the design cannot be safely simulated in this manner.

- -REPORT_X_capturing_cells { OFf | ON }

  An optional switch and literal pair that reports scan cells that capture X during simulation. When you set this option to ON, the tool displays this message when detecting a parallel pattern that captures X:

  ```
  // 10 scan cells capture X
  ```

  You can use "report_patterns -unload_x" to find the patterns that capture X on scan cells. In addition, by specifying the -pattern_index switch with -unload_x, report_patterns reports the scan cells that capture X on the specified pattern.

Example 1

The following example demonstrates disabling setting set/reset as the dominate synchronous port:

set_simulation_options -set_reset_dominate_port off

Example 2

The following example illustrates how to change the tool to model all _mux instances as non-consensus for both DRC checking and ATPG:

SETUP> **set_simulation_options -mux_select_x_sim_x on**

SETUP> **set_system_mode analysis**

```
// Flattening process completed, design_cells=28234 leaf_cells=1122 8 library_primitives=22323
sim_gates=45400 PIs=110 POs=103 CPU time=93.00 sec.

// ---------------------------------------------------------------------------

// Begin circuit learning analyses.

// ---------------------------------------------------------------------------

// Equivalent gates=54   classes=0   CPU time=9.00 sec.

…

ATPG> create_patterns
```

Related Commands

add_cell_constraints
set_clock_off_simulation
report_patterns

set_skewed_load

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

Mode: setup, analysis

Usage

set_skewed_load **OFf** | **ON**

Description

Specifies whether Tessent FastScan includes a skewed load in the patterns.

The set_skewed_load command either allows patterns to include a skewed load or restricts patterns from including a skewed load. You can use this command in any system mode, but if you use it outside the Setup mode, you must provide a skew_load procedure in the test procedure file. If the skew_load procedure does not exist, Tessent FastScan issues an error.

You use the skew_load procedure in LSSD designs to place different data in the master and slave of a scan cell by applying an additional clock pulse to the master shift clock.

Arguments

- **OFf**

  A literal that specifies to not include a skewed load in the patterns. This is the default behavior upon invocation of Tessent FastScan.

- **ON**

  A literal that specifies to include a skewed load in the patterns.

Example

The following example specifies for patterns to include the skewed load:

```
set_skewed_load on
set_system_mode analysis
create_patterns
```

set_split_capture_cycle

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

Mode: setup, analysis

Usage

set_split_capture_cycle **ON** | **OFf**

Description

Controls whether the tool updates simulation data between clock edges.

> **Tip**: If you use ATPG Expert (via the create_patterns command), you do not need to issue this command. ATPG Expert will automatically enable this command when it is needed. You may get the best results by letting ATPG Expert decide when to use this command.

The set_split_capture_cycle command enables or disables the simulation of level-sensitive and leading edge state elements updating as a result of applied clocks. When set to on, the tool updates simulation data between clock edges. This “split capture cycling” of data allows the tool to determine correct capture values for trailing edge and level-sensitive state elements, even in the presence of C3 and C4 violations. When set to “off”, simulation data is updated once each clock cycle.

Note that the set_split_capture_cycle command affects ATPG (subsequent create_patterns commands) but not logic simulation (subsequent simulate_patterns commands).

Arguments

- **ON**

  A literal that specifies to update ATPG data between clock edges. This is referred to as split-capture cycling.

- **OFf**

  A literal that that specifies to not update ATPG data between clock edges. This is the default behavior upon invocation of the tool.

Example

The following example enables split capture cycling:

```
set_split_capture_cycle on
```

Related Commands

set_clock_off_simulation

set_pattern_type

set_stability_check

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

Mode: setup, analysis

Usage

For dft -scan or when EDT is Off

set_stability_check {**ON** | **OFf** | **All_shift**}
[-Sim_static_atpg_constraints {OFf | ON}]
[-MAX_shift_cycles {*max_number* | DEfault}]

When EDT is On

set_stability_check {**ON** | **OFf**}
[-Sim_static_atpg_constraints {OFf | ON}]
[-MAX_shift_cycles {*max_number* | DEfault}]

Description

Specifies whether the tool checks the effect of applying the main shift procedure on non-scan cells.

In order to perform scan chain tracing, design rules checking (DRC) sometimes requires values to be present from non-scan state elements. For example, a control register in a test controller, such as a JTAG TAP, needs to be checked to see that the TAP register holds state during the shift procedure. By default, this checking is performed. This command controls the level of checking.

For detailed information about the use and reporting of state stability, refer to the "State Stability Reporting" appendix in the *Tessent Scan and ATPG User's Manual*.

Arguments

- **ON**

  A literal that enables the tool to perform a fast check of the effect of applying the main shift procedure on non-scan cells. This is the default behavior upon invocation of the tool.

- **OFf**

  A literal that disables the tool from performing any checks on the effect of applying the main shift procedure on non-scan cells.

- **All_shift**

  A literal that enables the tool to perform the most detailed level of checking. The main shift procedure is simulated for as many applications as the procedures call for. When you specify this option, it can significantly increase your run time.

Note

This option is primarily for logic BIST applications where the contents of shift registers need to be known at the end of scan loading, at the beginning of the capture cycle.

- -Sim_static_atpg_constraints OFf | ON

  An optional switch and literal pair that enables or disables the simulation of static ATPG constraints during DRC's state stability analysis. Static ATPG constraints are those you specify using the -Static switch with the add_atpg_constraints command.

  When you enable simulation of static ATPG constraints, the tool will assume the constraints are satisfied and will simulate their values during DRC's state stability analysis of nonscan memory elements. This simulation is useful in some clock gating situations.

  The invocation default is for DRC not to perform this simulation. The argument choices are as follows:

  OFf — A literal that disables the simulation of static ATPG constraints during DRC. This is the invocation default.

  ON — A literal that enables the simulation of static ATPG constraints during DRC.

- -MAX_shift_cycles *max_number* | DEfault

  An optional switch and integer or literal pair that limits the number of shift cycles. You can only use this switch when stability check is set to ON (set_stability_check ON).

  The *max_number* is an integer greater than or equal to 1 (one) that limits the maximum number of shift cycles the state stability analysis can simulate. When setting *max_number*, you should follow the guidance of the D12 DRC warning message for the number of shift cycles; the tool automatically considers existing independent shift cycles when calculating this recommended number.

  Use the DEfault option with this switch to revert to the default behavior of state stability analysis without imposing a specific shift cycle limit using the following syntax:

  set_stability_check ON -MAX_shift_cycles default

Example 1

The following example enables fast stability checking of the effect of applying the main shift procedure on non-scan cells:

set_stability_check on

Example 2

The following example enables fast stability checking as well as the simulation of static ATPG constraints during DRC. Notice the use of the report_environment command to confirm the resultant changes in tool settings:

report_environment

```
...
split capture cycle =     OFF
stability check =         OFF
...
```

set_stability_check on -sim_static_atpg_constraints on
report_environment

```
...
split capture cycle =     ON
stability check =         ON
...
```

set_static_learning

Context: unspecified, all contexts

Mode: setup

Usage

set_static_learning [ON | OFf] [-Limit *integer*] [-Verbose { ON | OFf }]

Description

Specifies whether the tool performs the learning analysis to make the ATPG process more efficient.

The set_static_learning command controls whether the tool performs the learning analysis immediately after design flattening. The tools use the learned behavior for intelligent decision making in later processes, such as ATPG and DRC.

When static learning analysis is on, the additional learned behavior focuses primarily on bus gates. Static learning analysis also allows the test pattern generation process to immediately recognize conflicts and restricted decisions on ATPG constraints that result from the gate assignments. By enabling static learning, you prevent the tools from unnecessarily remaking many decisions, thereby improving the ATPG performance.

For more information about the learning analysis, refer to "Learning Analysis" in the *Tessent Scan and ATPG User's Manual*.

Arguments

- ON | OFf

  An optional literal that disables additional static learning analysis. The default enables the tool to perform additional learning analysis, but you may want to disable to save time if you are not going to be running ATPG.

- -Limit *integer*

  An optional switch and integer pair that specifies a single gate simulation activity threshold. When the tool reaches that threshold, it discontinues learning on gates in that design region. You would use the -Limit switch for performance reasons. The default value for the *integer* option is 1000.

Caution

While changing the learning limit increases performance for some designs, it significantly decreases performance for a vast majority of designs. It is very unusual to need to change the learning limit and is therefore not recommended.

- -Verbose { ON | OFf }

  An optional switch and literal pair that enables additional detailed reporting of analysis results, including the numbers of equivalent gates, classes, implications, and tied gates.

Example

The following example first enables access to the learned information and then enables the tool to perform additional learning analysis before design flattening:

```
set_learn_report on
set_static_learning on -limit 500
set_system_mode analysis
```

Related Commands

set_learn_report

set_system_mode

Context: all contexts

Mode: all modes

Usage

set_system_mode {**setup** | **insertion** | **analysis**} [-force]

Description

Specifies the operational state you want the tool to enter.

The default system mode after tool invocation is setup. The context the tool is in determines which system mode you can specify with the set_system_mode command. See the *Tessent Shell User's Manual* for more information about contexts and system modes.

Arguments

- **setup**

  A required literal that specifies to enter Tessent Shell setup mode.

- **insertion**

  A required literal that specifies to enter Tessent Shell insertion mode.

- **analysis**

  A required literal that specifies to enter Tessent Shell analysis mode.

- -Force

  An optional switch that suppresses the errors that are normally generated when exiting analysis or insertion system modes with unsaved data. Instead of issuing an error, the tool silently deletes unsaved data.

  You can also use the -force switch in the following situations:

  - When you are in the "patterns -ijtag" context and try to move out of analysis system mode without saving your pattern sets, you cannot change the system mode until you either save the pattern sets or use the -force switch to discard all unsaved pattern sets.
  - When you are in insertion system mode and in the rtl context, if you try to change to setup system mode without first saving edited or created modules, you cannot change the system mode until you either save the modules or use the -force switch to discard the unsaved modules.

Example

The following example changes the system mode so that you can perform an insertion run:

```
set_context dft -no_rtl
read_verilog mydesign.v
read_cell_library atpg.lib
set_current_design
set_system_mode insertion
create_instance u1/myInst -of_module myModule
```

Example

The following example changes the system mode so you can perform an ATPG run:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 indata2 outdata4
set_system_mode analysis
create_patterns
```

Example2

The following example changes the system mode so that you can perform scan stitching:

```
add_tied_signals 1 vcc
add_tied_signals 0 vss
add_clocks 0 clock
set_system_mode analysis
report_sequential_instances
insert_test_logic
```

Related Commands

get_system_mode

set_test_end_icall

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

Mode: all modes

Usage

set_test_end_icall {[icl_instance_path.]**iproc_name** [arguments …]} [[-merge] -append]

Description

Adds an iCall to the start of the test_end procedure.

The set_test_end_icall command adds an iCall to the start of the test_end procedure.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

If you specify the –append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order they were appended.

If you issue the command without the -append switch, it overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

Arguments

- {[icl_instance_path.]**iproc_name** [arguments …]}

  The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- -append

  An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.

- -merge

  An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the iMerge command and cannot be unspecified.

  Refer to iMerge Conflict Reporting in the *Tessent IJTAG User's Manual* for more information on the -error_on_conflict option.

  The merged iCall commands can be specified in the test_setup/test_end procedures using the tessent shell commands set_test_setup_icall/set_test_end_icall. Later, when you use the tessent shell command report_procedures (available in analysis mode), the merged iCall commands of the test_setup/test_end procedures will appear as follows:

```
iMerge
        iCall iProcName iProcArgs
        ....
iMerge -end;
```

The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

Example

In the following example, the iCall commands specified using the -merge option in the test_end file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_end_icall iProc1 -append
set_test_end_icall iProc2 -append
set_test_end_icall iProc3 -merge -append
set_test_end_icall iProc4 -merge -append
```

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_end =
        iCall iProc1;
        iMerge =
            iCall iProc2;
            iCall iProc3;
            iCall iProc4;
        end;
end;
```

Related Commands

get_test_end_icall_list
get_test_setup_icall_list
iCall
set_test_setup_icall

set_test_logic

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_test_logic {**-Set** {**ON** | **OFf**} | **-REset** {**ON** | **OFf**} | **-Clock** {**ON** | **OFf**} | **-RAm** {**ON** | **OFf**} | **-C6 {ON | OFf}**}…

Description

Inserts test logic to control the set, reset, clock, enable, or write control signals to make them scannable when scan chains are inserted.

Before inserting scan chains, you must specify the cell models with the add_cell_models command. You can use the report_dft_check command to display the signals where test logic can be inserted to make the signals controllable.

By default, no test logic is inserted on control signals. You can use the report_environment command to display the current test logic settings.

If all enable signals of a bus are driven purely by combinational logic, no additional test logic is needed to prevent contention due to scan shifting.

Arguments

- **-Set ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make set signals controllable. The default setting is off.

- **-REset ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make set signals controllable. The default setting is off.

- **-Clock ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make clock signals controllable. The default setting is off.

- **-RAm ON | OFf**

  A switch and literal pair that enables the insertion of test logic to make read and write signals controllable. The default setting is off.

  This option does not affect the set and reset signals of the RAM. You should use the *edge_trigger* attribute in the library definition of the RAM to specify inserting test logic to control these signals. For more information, see the "Attributes of RAM/ROM Primitives" section of the *Tessent Cell Library Manual*.

- **-C6 {ON | OFf}**

  A switch and literal pair that, when enabled, instructs the tool to add hardware to *fix* identified C6 violations. The tool fixes a C6 violation by inserting a multiplexer into the data path that feeds the flip-flop generating the violation. The test mode input of the multiplexer is driven by a nearby scan flip-flop. When a top-level clock feeds the data input of many flip-flops (which results in multiple C6 violations), the multiplexer is inserted at the highest possible level of hierarchy. The driving scan cells are selected to be in the same clock domain as the flip-flops generating the C6 violations. This setting is off by default.

Note

This option cannot be used when the “set_drc_handling -Conservative” switch is enabled.

Example

The following example checks the set and clock signals of uncontrollable memory elements and makes them controllable with the addition of test logic:

```
add_clocks 0 clk
set_test_logic -set on -clock on
set_system_mode analysis
report_dft_check
add_cell_models and2 -type and
add_cell_models or2 -type or
add_cell_models mux21h -type mux s a b
add_cell_models nor2 -type nor
report_cell_models
insert_test_logic
```

Related Commands

add_cell_models
delete_cell_models
report_cell_models
set_latch_handling

set_test_point_analysis_options

Context: dft -test_points

Mode: setup, analysis

Usage

set_test_point_analysis_options
[-total_number ***number*** | -control_points ***number*** -observe_points ***number***]
[-test_coverage_target ***percentage***]
[-pattern_count_target ***number***]
[-max_control_points_per_path ***number***]
[-exclude_cross_domain_paths **on** | **off**]

Description

Sets the maximum number of test points, the breakdown in control and observe points, the target fault coverage and the number of pseudo random patterns to be applied and some other parameters that are taken into account during test point analysis.

Not all gates in the design will be considered by the tool for inserting test points. For various reasons, test points will not be inserted on gates in the clock tree or in the scan path, gates in protected blocks or gates where the user has explicitly requested that no test points should be inserted. If there are many locations where test points cannot be inserted the tool may not be able to find the best locations for test points. This may increase the number of test points that are needed to reach the fault coverage target. Therefore, if test points cannot be inserted at more than 20% of the gate-pins the tool will generate a warning. The warning specifies the number of these locations as well as the main reason(s) why test points cannot be inserted at these locations.

Please refer to the appendix "Test Point Analysis for Target Faults" in the *TK/LBIST Flow User's Manual* for generating test points that improve the testability of a subset of faults.

Arguments

- -total_number ***number***

  An optional switch and integer that specifies the maximum number of test points to be inserted with no maximum as default. This maximum only applies to the test points automatically generated by the tool using the command analyze_test_points.

- -control_points ***number***

  An optional switch and integer that specifies the maximum number of control points to be inserted.

  This maximum only applies to the control points automatically generated by the tool using the analyze_test_points command. Issuing these two commands is sub-optimal; the preferred flow is to let the tool automatically decide the test point distribution.

- -observe_points ***number***

  An optional switch and integer that specifies the maximum number of observe points to be inserted.

  This maximum only applies to the observe points automatically generated by the tool using the analyze_test_points command. Issuing these two commands is suboptimal; the preferred flow is to let the tool automatically decide the test point distribution.

- -test_coverage_target ***percentage***

  An optional switch and integer that specifies the target fault coverage to be achieved by the number of random patterns. The tool will insert test points until either the maximum number of test points is reached or the desired fault coverage with the target number of random patterns is achieved. The default is 99%.

- -pattern_count_target ***number***

  An optional switch and integer that specifies the number of random patterns to be applied to the design. Fault coverage estimations will be done using this parameter as reference. Default is 100,000.

- -max_control_points_per_path ***number***

  An optional switch and integer that specifies the maximum number of control points to be inserted on a single path. If it is not specified, no maximum is the default.

- -exclude_cross_domain_paths **on** | **off**

  An optional switch and literal that prevents the tool from inserting test points in inter-clock domains. The default is off.

Example 1

The following example specifies that enough test points should be generated to achieve 99.5 percent fault coverage, but not more than a maximum number of 3000 test points, for a design that will be tested with a set of 131072 random patterns:

**set_test_point_analysis_options -test_coverage_target 99.5 -total 3000 **
-pattern_count_target 131072

The following is a snippet from the log file that shows the test coverage reporting before and after selecting test points:

```
//  command: analyze_test_points
//  Identifying locations of x-bounding muxes prior to test point analysis
//
//  Test Coverage Report before Test Point Analysis
//  -----------------------------------------------
//  Target number of random patterns           131072
//
//  Total Number of Faults                      15577
//    Testable Faults                           15063  (  96.70%)
//      Logic Bist Testable                     14970  (  96.10%)
//      Blocked by xbounding                        0  (   0.00%)
//      Uncontrollable/Unobservable                93  (   0.60%)
```

```
//
//  Estimated Maximum Test Coverage                        99.38%
//  Estimated Test Coverage (pre test points)              96.19%
//  Estimated Relevant Test Coverage (pre test points)     96.24%
//
//  Warning: Test points cannot be inserted at 4033 (33.6%) gate-pins.
// This may increase the number of test points that is needed to reach the
fault coverage target.
// 3929 (32.7%) gate-pins are located on clock lines or on the scan path.
//
//  Incremental Relevant Test Coverage Report
//  -----------------------------------------
//  TPs   50 =   21 (CP) +   29 (OP), TC 99.41
//  TPs  100 =   42 (CP) +   58 (OP), TC 99.41
//  TPs  150 =   62 (CP) +   88 (OP), TC 99.42
(snipped)
//  TPs 1150 =  128 (CP) + 1022 (OP), TC 99.49
//  TPs 1200 =  135 (CP) + 1065 (OP), TC 99.49
//  TPs 1250 =  148 (CP) + 1102 (OP), TC 99.50
//
//  Test point analysis completed: target estimated test coverage has been
achieved.
//  Inserted Test Points                          1297
//    Control Points                               157
//    Observe Points                              1140
//
//
//  Test Coverage Report after Test Point Analysis
//  ----------------------------------------------
//  Target number of random patterns            131072
//
//  Total Number of Faults                       18042
//    Testable Faults                            17528  (  97.15%)
//      Logic Bist Testable                      17435  (  96.64%)
//      Blocked by xbounding                         0  (   0.00%)
//      Uncontrollable/Unobservable                 93  (   0.52%)
//
//  Estimated Test Coverage (post test points)             99.03%
//  Estimated Relevant Test Coverage (post test points)    99.50%
```

Note

The “Total Number of Faults”, the number of “Testable Faults” and the number of “Logic Bist Testable” faults are larger after test point insertion as these numbers include the (testable) faults at the new test point logic.

Example 2

The following example specifies that enough test points should be generated to achieve 99 percent fault coverage (default), but not more than a maximum number of 3000 control points, and a maximum of 2000 observe points:

set_system_mode analysis

set_test_point_analysis_options -control_points 3000 -observe_points 2000

analyze_test_points

report_test_points

insert_test_logic

Related Commands

analyze_test_points

report_test_points

set_test_point_insertion_options

set_test_point_insertion_options

Context: dft -test_points

Mode: setup, analysis

Prerequisites: None

Usage

set_test_point_insertion_options
[-control_point_enable ***pin_path_name***]
[-observe_point_enable ***pin_path_name***]
[-observe_point_share ***integer***]
[-test_point_clock **on** | **off**]

Description

Sets the following test point insertion related parameters:
- name of the test point enable signals.
- number of test points that will be observed using a single flip-flop.
- connect the clock port of all newly inserted flip-flops to a global test_clock signal instead of re-using existing clock signals.

These parameters will impact the structure of the test logic that is generated when the insert_test_logic command is issued.

Arguments

- -control_point_enable ***pin_path_name***

  An optional switch and string pair that specifies the pin and path name of the enable signal for control points. In its absence, the name lbist_enable will be used at the top level.

- -observe_point_enable ***pin_path_name***

  An optional switch and string pair that specifies the pin and path name of the enable for observe points. In its absence, the name lbist_enable will be used at the top level.

- -observe_point_share ***integer***

  An optional switch and integer pair that specifies how many observe points will be observed by one scan cell. The default is 16 observe points to one destination scan cell.

- -test_point_clock **on** | **off**

  An optional switch and literal that specifies using a global clock to drive scan cells for test points. The default is off. When set to on, the tool takes the global clock name from the -tclk option of the set_scan_insertion command.

Example 1

The following example uses the “-observe_point_share” switch to determine the size of the XOR tree that is generated when the tool merges several observe points together before

capturing the value into either a new or existing scan cell. This example XOR's together no more than 7 observe points before feeding the output of the XOR tree into a scan cell.

```
set_test_point_insertion_options -observe_point_share 7
```

Example 2

The following example specifies the names of enables for control points and observe points respectively. It also specifies that the global clock that was specified with the set_scan_insertion command ('test_clk') is to be used to drive the scan cells for the test points:

```
set_scan_insertion -tclk test_clk
...
set_test_point_insertion_options -control_point_enable  control_enable
set_test_point_insertion_options -observe_point_enable  observe_enable
set_test_point_insertion_options  -test_point_clock on
```

Example 3

The following example generates a maximum of 5000 test points. Then it specifies the names of enables for control points and observe points respectively, and it inserts the test points into the netlist:

```
set_system_mode analysis
set_test_point_analysis -total_number 5000
analyze_test_points
report_test_points
set_test_point_insertion_options -control_point_enable cp_enable \
        -observe_point_enable op_enable
insert_test_logic
```

Related Commands

add_cell_models

insert_test_logic

set_test_setup_icall

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

Mode: all modes

Usage

set_test_setup_icall {[icl_instance_path.]**iproc_name** [arguments …]} [[-merge] -append]

Description

Adds an iCall to the end of the test_setup procedure.

The set_test_setup_icall command adds an iCall to the end of the test_setup procedure.

The command accepts a Tcl list consisting of the three components that normally follow an iCall: the name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

If you specify the –append switch, the new iCall is appended to the iCalls in memory. Each iCall with its arguments is called in the order they were appended.

If you issue the command without the -append switch, it overwrites the previous values. To clear the list, assign an empty string or empty list without the -append switch.

The -merge switch provides support for iMerge in the test procedure file. It can only be used if the -append switch is also specified.

Arguments

- {[icl_instance_path.]**iproc_name** [arguments …]}

  The name of the iProc, preceded by the optional ICL instance path and separated by a dot, and one or more optional arguments to the iProc.

- -append

  An optional switch that appends the specified list of iCalls to the accumulated list of iCall commands. Note that if you issue the command without the -append switch, the accumulated iCalls are cleared and only the current specified iCall command is stored.

- -merge

  An optional switch to merge iCall commands together. The -merge option can only be used with the -append switch to merge the specified iCall with the previous iCall. All the merged iCall commands infer the -error_on_conflict option of the iMerge command and cannot be unspecified.

  Refer to iMerge Conflict Reporting in the *Tessent IJTAG User's Manual* for more information on the -error_on_conflict option.

  The merged iCall commands can be specified in the test_setup/test_end procedures using the tessent shell commands set_test_setup_icall/set_test_end_icall. Later, when you use the tessent shell command report_procedures (available in analysis mode), the merged iCall commands of the test_setup/test_end procedures will appear as follows:

```
iMerge
        iCall iProcName iProcArgs
        ....
iMerge -end;
```

The -error_on_conflict option is always enabled, so it is not necessary to specify it explicitly.

Example 1

The following example adds an iCall to the end of the test_setup procedure, then appends and merges a second iCall. The get_test_setup_icall_list command displays the added iCalls.

```
set_test_setup_icall "m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on
        edt_bypass off"

set_test_setup_icall "m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on
        edt_bypass off" -append -merge

get_test_setup_icall_list
```

```
{m8051_m8051_B1_edt_i.setup edt_low_power_shift_en on edt_bypass off}
{m8051_m8051_B2_edt_i.setup edt_low_power_shift_en on edt_bypass off}
```

Example 2

In the following example, the iCall commands specified using the -merge option in the test_setup file merge iProc2, iProc3 and iProc4 into iProc1:

```
set_test_setup_icall iProc1 -append
set_test_setup_icall iProc2 -append
set_test_setup_icall iProc3 -merge -append
set_test_setup_icall iProc4 -merge -append
```

These commands are equivalent to the sequence of the iCall commands in the test procedure file:

```
Procedure test_setup =
        iCall iProc1;
        iMerge =
            iCall iProc2;
            iCall iProc3;
            iCall iProc4;
        end;
end;
```

Related Commands

get_test_end_icall_list
get_test_setup_icall_list
iCall
set_test_end_icall

set_tied_signals

Context: all contexts

Mode: setup

Usage

set_tied_signals **X** | **1** | **0** | **Z**

Description

Changes the default value for floating pins and floating nets which do not have assigned values.

The set_tied_signals command specifies the default value that the tool ties to all floating nets and floating pins that you do not specify with the add_tied_signals command. Upon invocation of the tool, if you do not assign a specific value, the tool assumes the default value is unknown (X).

If the model is already flattened and then you use this command, you must delete and recreate the flattened model.

Arguments

- **X**

  A literal that ties the floating nets or pins to unknown. This is the default upon invocation of the tool.

- **0**

  A literal that ties the floating nets or pins to logic 0 (low to ground).

- **1**

  A literal that ties the floating nets or pins to logic 1 (high to voltage source).

- **Z**

  A literal that ties the floating nets or pins to high-impedance.

Example

The following example ties floating net vcc to logic 1, ties the remaining unspecified floating nets and pins to logic 0, then performs an ATPG run:

```
set_tied_signals 0
add_tied_signals 1 vcc
set_system_mode analysis
create_patterns
```

Related Commands

add_tied_signals
delete_tied_signals
report_tied_signals
Set Floating Net_type

set_timing_exceptions_handling

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

Mode: setup, analysis

Usage

set_timing_exceptions_handling [ON | OFf] [-CLOCK_groups { ON | OFf }]
[-ADD_Hold_time_when_no_settings_in_sdc {ON | OFf }] [-LIne_no {ON | OFf}]
[-SETUp { ON | OFf }]
[-APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFf}]
[-DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFf}]

Description

Changes the default handling of timing exception paths (false paths and multicycle paths) read by the read_sdc command.

When you issue the set_timing_exceptions_handling command without options, the command reports the current settings.

Arguments

- ON | OFf

  A literal that enables or disables handling of timing exceptions on false paths and multicycle paths. This capability is enabled by default upon invocation of the tool. Turning off this switch is useful for debugging low coverage or simulation mismatch because you can quickly verify if the masking values or test coverage loss is due to timing exceptions. Turning off this switch can result in pattern mismatches, so you should not turn off this switch other than for debugging purposes.

- -CLOCK_groups { ON | OFf }

  A switch and literal pair that allows you to change the clock_groups handling in SDC. By default, the tool expands timing exceptions for the clocks in asynchronous clock groups as both setup and hold timing exceptions. However, this can create large memory and run-time overhead. If you are not generating any patterns that pulse asynchronous clock domains during ATPG, you can disable this path expansion by specifying Off.

- -ADD_Hold_time_when_no_settings_in_sdc {ON | OFf }

  A switch and literal pair that allows you to treat the timing exceptions to include both setup and hold-time violations when the SDC files specifies neither –setup nor –hold. By default, the tool matches the SDC file used by your static timing analyzer, which is useful if your design must include hold timing exceptions during stuck-at ATPG. When you turn off this switch, the tool assumes that all false paths without -setup or -hold options have only setup timing exceptions.

Table 7-52 further explains how the tool treats timing exception paths based on the setting in the SDC file and the -ADD_Hold_time_when_no_settings_in_sdc switch:

Table 7-52. SDC File and -ADD_Hold_time_when_no_settings_in_sdc switch

SDC file switches	Default tool behavior	Tool behavior with switch OFF
none	setup & hold	setup *
-setup	setup	setup
-hold	hold	hold
-setup -hold	setup & hold	setup & hold

* True unless the timing exception path is added across clock domains, in which case the path is treated as both setup & hold. As an example, the following SDC commands are equivalent:
`set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}}`
`set_false_path -from {get_clocks {CLK1}} -to {get_clocks {CLK2}} -setup -hold`		

- -LIne_no {ON | OFf}

  An optional switch and literal pair that allows you to turn on reporting of SDC filename and line numbers for timing violations when using the report_false_paths or the report_multicycle_paths command. By default, the tool does not report the SDC filename and line numbers.

- -SETUp { ON | OFf }

  An optional switch and literal pair that disables setup time simulation without having to modify the SDC file. By default, setup time simulation is enabled.

  Note that the tool automatically reclassifies AU faults when you change this option, so there is no need to issue a “reset_au_faults -reclassify” command.

- -APPLY_Clock_scaling {WIthin_each_source_clock | ACross_all_source_clocks | OFf}

  An optional switch and literal pair that specifies the behavior of multicycle path (MCP) cycle scaling.

  The following text explains the concept of cycle scaling. For synchronous clock domains with different frequencies (either within each source clock or across all source clocks), the tool scales the clocks according to their frequency information provided in the SDC file. In cases when the driving clock of a multicycle path has a different frequency source clock, this switch defines how the tool should determine their relationship. For example, consider a multicycle path with 2 cycles defined with -end in the SDC file. If the frequency of the end-point driving clock is twice as slow as its source clock, the tool scales the clocking so that the path is treated as a multicycle path with 4 cycles.

  This switch has the following options:

  WIthin_each_source_clock — Applies the cycle scaling only for the generated clocks defined in SDC using the create_generated_clock command with the same source clock. Treats different source clocks independently, and treats all generated clocks with the same source clock as synchronous clocks. This is the default.

ACross_all_source_clocks — Applies the cycle scaling across all source clocks. This option is applicable when treating all source clocks as synchronous.

OFf — Does not apply the cycle scaling, so the tool uses the cycle count as defined in SDC set_multicycle_path commands.

- -DERIVE_Fault_detection_cycles_from_SDC_generated_clocks {ON | OFf}

  An optional switch and literal pair that allows the tool to ignore SDC-generated clocks from which to derive the fault detection cycles. By default, the tool allows ATPG and the fault simulator to consider the transition faults in slower clock domains to be detected with multiple activation cycles without needing to define multicycle paths. For example, if a transition fault is on a divide-by-2 clock domain, the fault can be activated for up to 2 cycles before capture. The switch default is ON, which instructs ATPG and the fault simulator to consider the transition faults activated up to *n* cycles for divide-by-*n* clock domains unless explicitly specified using multicycle paths. Note that the tool takes into account the SDC command “create_generated_clock -divide_by” but only in the absence of the -multiply_by switch. The SDC command “create_generated_clock -multiply_by” is unsupported.

Tip: You can use the report_clocks command to list all of the SDC-defined clocks.

Example 1

The following example specifies that the tool not expand timing exception paths for the clocks in asynchronous clock groups as both setup and hold timing exception paths.

```
set_timing_exceptions_handling -clock_groups off
```

Example 2

The following example shows how to specify reporting of SDC filename and line numbers for timing violations:

```
read_sdc timing.sdc
set_timing_exceptions_handling -line_no on
report_false_paths

// False Path -from_clock cmem_fclki cmem_fclkin -to_clock p_clk spm_pci_clk -setup (filename = \
   timing.sdc, line 406)

// False Path -from_clock p_clk spm_pci_clk -to_clock cmem_fclki cmem_fclkin -setup (filename = \
   timing.sdc, line 408)
...

// Total reported false paths = 177
```

Related Commands

read_sdc
report_clocks
report_false_paths
report_multicycle_paths

set_tla_loop_handling

Context: all contexts

Mode: setup

Usage

set_tla_loop_handling [ON | OFf]

Description

Specifies how to simulate feedback loops that contain transparent latches (TLAs).

By default, TLAs in feedback loops are simulated using the same iterative algorithm used to simulate loops through combinational logic. This can increase test coverage; however, it requires feedback loop analysis to be performed whenever DRC is run, which may result in a slightly longer DRC run time. When this command is disabled, feedback loop analysis is performed only after flattening; however, TLAs that are in feedback loops are treated as TIEX gates during simulation, which can reduce test coverage.

Note

Be aware that extra processing time is required to support loop simulation through TLAs. In some circumstances, the improved fault detection yields sufficient reduction in ATPG effort that overall runtimes decrease, but you should anticipate an increase in runtimes in most cases.

Any TLA that is in a feedback loop is treated as a place to break the loop (similar to the FB buffer gate used for combinational loops). Strongly connected sets of gates are identified as feedback networks. You can display feedback networks by using the report_feedback_paths command.

You can use the -Environment switch of the write_patterns command to save the value of the -Iteration switch. You can use the report_environment command to see the current value of the set_tla_loop_handling command.

Arguments

- ON

  An optional literal that enables TLA loop simulation. This is the default.

- OFf

  An optional literal that disables TLA loop simulation.

Related Commands

report_environment
report_feedback_paths
write_patterns
set_loop_handling

set_trace_flat_model_options

Context: all contexts

Mode: setup, analysis

Usage

set_trace_flat_model_options [-tag_condition *tag_attribute_expression*]
[-stop_condition *stop_attribute_expression*]
[-controllability {controlling | constant | unblocked | connected}]
[-stop_at_first_tag {ON | OFf}] [-max_levels {unlimited | *number*}]
[-latch {normal | transparent}] [-map_tag_to_design_module_boundary { ON | OFf }]

Description

Modifies the default behavior of the trace_flat_model command.

This command sets options as global defaults. These defaults will always be applied if they are not specified in the current trace_flat_model execution.

This command removes the requirement for you to repeatedly specify trace configuration information for each call to the trace_flat_model command.

Arguments

- -tag_condition *tag_attribute_expression*

  An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the -filter option used for the get_pins command. See section "Attribute Filtering Equation Syntax" for more details on filtering attribute equation formats.

- -stop_condition *stop_attribute_expression*

  An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the gate_pin failing the specified controllability requirements. The pins of memory or dff primitives are always stop points even when not explicitly specified.

- -controllability {controlling | constant | unblocked | connected}

  An optional switch and literal pair that defines the controllability requirements before traversing a gate.

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output without an inversion,

  constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

connected—With this value, gates are always traversed when reached.

- -stop_at_first_tag {ON | OFf}

  An optional switch and literal pair that specifies whether to interrupt tracing when the first tag object with the specified controllability is found. This is useful when checking for at least one occurrence of a given condition in the fanin or fanout of a pin without wasting time to find them all.

- max_levels {unlimited | *number*}

  An optional switch and value pair that defines the maximum level of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count.

- -latch {normal | transparent}

  An optional switch and literal pair that specifies the behavior of tracing through a latch.

  normal — A latch is traced through only if the simulated value in stable_after_setup makes the latch satisfy the specified controllability requirements.

  transparent — All latches are traced through regardless of the value at the latch enable port.

- -map_tag_to_design_module_boundary { ON | OFf }

  An optional switch and literal pair that specifies whether to map the tag points to gate_pins outside of library cells. When this switch is ON, the trace_flat_model command maps the tag points to gate_pins outside of library cells and never inside the cells. Library cells can be either cells from a ATPG library or `celldefine Verilog modules.

Example 1

The following example script shows how the set_trace_flat_model_options command uses attribute expressions to constrain the trace for level 1 clock gaters.

```
set stop_condition_expr " {isClockGater == true} || {isRegister == true} "
set tag_condition_expr "isClockGater == true"

set_trace_flat_model_options -tag_condition  ${tag_condition_expr}  \
   -stop_condition ${stop_condition_expr}

foreach_in_collection pin [all_clocks] {
    set CGs [trace_flat_model -from $pin -direction forward]
        if {[sizeof_collection $CGs] > 0} {
            puts "Level1 clock gaters for clock ${pin} are:"
            foreach_in_collection CG ${CGs} {
                puts "  ${CG}"
            }
        }
    }
}
```

Related Commands

get_trace_flat_model_option

trace_flat_model

set_trace_report

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

Mode: setup, analysis

Usage

set_trace_report **OFf** | **ON**

Description

Specifies whether the tool displays gates in the scan chain trace.

The set_trace_report command controls whether the tool displays all of the gates in the scan chain trace during rules checking.

Arguments

- **OFf**

  A literal that specifies for the tool not to display gates in the scan chain trace. This is the default behavior upon invocation of the tool.

- **ON**

  A literal that specifies for the tool to display gates in the scan chain trace during rules checking.

Example 1

The following example displays the gates in the scan chain trace during rules checking:

```
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Example 2

The following example displays the gates in the scan chain trace during rules checking:

```
add_clocks 0 clock
add_scan_groups group1 scanfile
add_scan_chains chain1 group1 indata2 outdata4
set_trace_report on
set_system_mode analysis
```

Related Commands

add_scan_chains

report_scan_chains

set_transcript_style

Context: unspecified, all contexts

Mode: all modes

Usage

set_transcript_style {**off** | **full** | **tool_commands_only** | **input_only**}

Description

Specifies the style in which the tool transcripts commands.

Arguments

- **off**

  A keyword that turns off all command transcription so that the commands run as though included using the Tcl “source” command. This option allows you to execute scripts that don’t echo the tool commands, which can be helpful running scripts that use the report commands as required for introspection.

- **full**

  A keyword that transcripts all commands read from a dofile before any Tcl evaluation is done. This includes all Tcl commands and constructs.

  Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs will transcript as a single item. In addition, when non-introspection tool commands are embedded within another command, they transcript as they are about to execute. (Introspection commands are commands that simply return information about the design or tool state and therefore have no effect on operation of the tool. Tool commands included within the catch_output command are also considered introspection commands since they do not write to the transcript.)

  As in a standard Tcl shell, any result returned from each top-level Tcl command transcripts when you run the tool in interactive mode. In batch mode, this option does not transcript command results.

- **tool_commands_only**

  A keyword that transcripts tool commands, including all arguments, as the commands execute. The command string is transcribed after being evaluated by Tcl so all variable and command references are resolved. For commands in a loop or if/else construct, the tool transcripts each execution of each command plus any output produced so that the context is evident.

  This option does not transcript Tcl flow control constructs, pure Tcl commands, and introspection commands nor their results. This option essentially specifies the legacy behavior of the tools.

- **input_only**

  A keyword that transcripts all input from the dofile (both Tcl and tool commands) before sending to Tcl for evaluation. Since the command interpreter accumulates input until receiving a complete Tcl command before sending to Tcl for evaluation, entire if/else, for, and foreach constructs transcript as a single item. Embedded tool commands are not transcribed again when they execute.

Example

The following example consists of a dofile and the transcription results of the various style options.

Dofile:

```
set foo 2
set myclk(1) CLK1
set myclk(2) CLK2
set pfile mytest.proc
if { $foo == 2 } {
   add_clocks 0 $myclk(1)
   add_clocks 0 $myclk(2)
}
add_scan_groups g1 $pfile
```

Full-style transcript:

```
//  command: set foo 2
//  command: set myclk(1) CLK1
//  command: set myclk(2) CLK2
//  command: set pfile mytest.proc
//  command: if { $foo == 2 } {
//                     add_clocks 0 $myclk(1)
//                     add_clocks 0 $myclk(2)
//                   }
//  sub-command: add_clocks 0 CLK1
//  sub-command: add_clocks 0 CLK2
//  command: add_scan_groups g1 $pfile
```

Tool_commands_only-style transcript:

```
//  command: add_clocks 0 CLK1
//  command: add_clocks 0 CLK2
//  command: add_scan_groups g1 mytest.proc
```

Input_only-style transcript:

```
//  command: set foo 2
//  command: set myclk(1) CLK1
//  command: set myclk(2) CLK2
//  command: set pfile mytest.proc
//  command: if { $foo == 2 } {
//                     add_clocks 0 $myclk(1)
//                     add_clocks 0 $myclk(2)
//                   }
//  command: add_scan_groups g1 $pfile
```

Related Commands

get_transcript_style

set_logfile_handling

set_transient_detection

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

Mode: setup, analysis (dft -scan context only)

Usage

set_transient_detection **OFf** | {**ON** [-Verbose | -NOVerbose]}

Description

Specifies whether the tool detects all zero width events on the clock lines of state elements.

The set_transient_detection command sets the simulator to detect all zero width events on the clock lines of state elements. If the zero width event causes a change of state in the state element, the tool sets that state element to X.

If Off is specified, DRC simulation treats all events on state elements as valid. Because the simulator is a zero-delay simulator, it is possible for DRC to simulate zero width, monostable circuits with ideal behavior which is rarely matched in silicon. The resulting zero width output pulse from the monostable circuit is also treated as a valid clocking event for other state elements.

Arguments

- **OFf**

  A literal that specifies for the tool to set transient detection off.

  Caution

  This argument is only for special cases where you understand the potential hazards from transients on the clock lines and know how to handle them. Turning transient detection off without taking care of the hazards can result in bad patterns that mismatch when verified in a timing based simulator. Be sure you understand the consequences and are willing to accept the risks before turning transient detection off.

- **ON**

  A literal that specifies for the tool to set transient detection on. This is the invocation default.

- -Verbose

  An optional switch that specifies for the tool to display a message identifying each time a state element is set to X due to transient detection. This is the invocation default and is valid only when transient detection is enabled.

- -NOVerbose

  An optional switch which specifies for the tool *not* to display a message identifying each time a state element is set to X due to transient detection. This switch is valid only when transient detection is enabled.

Example

Transient detection is enabled in verbose mode by default. Unless you use the set_transient_detection command to modify the default, you will see a message similar to the following for each transition that is caused by a transient, or zero-width, pulse:

```
//  Warning: 0 width pulse detected on pin 2 of
/my_design/my_inst/my_latch/ (463228) in procedure my_procedure at time 0.
```

You can eliminate these messages, while retaining the advantages of transient detection, by issuing this command:

set_transient_detection on -noverbose

Or you can disable transient detection, causing DRC simulation to treat all events on state elements as valid.

set_transient_detection off

set_transition_holdpi

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

Mode: setup, analysis

Usage

set_transition_holdpi **ON | OFf**

Description

Specifies for the tool to freeze all primary input values other than clocks and RAM controls during multiple cycles of pattern generation.

The set_transition_holdpi command lets you turn this feature on while the fault type is in “transition”. This is useful in cases where it is not practical to maintain high data rates to the primary input pins of the device under test on a tester. Primary inputs may still change from pattern to pattern however.

Note

Because launch off shift transition patterns require the scan enable signal (which is not a clock or RAM control) to transition within a single pattern, the tools ignore this command when generating this type of pattern.

Arguments

- **ON**

  A literal that specifies for the tool to hold all primary input values (other than clocks and RAM controls) during multiple cycles of pattern generation.

- **OFf**

  A literal that specifies for the tool not to hold all primary input values. This is the invocation default.

Example

The following example holds all primary input values.

```
set_transition_holdpi on
```

Related Commands

report_primary_inputs

set_pathdelay_holdpi

set_tristate_gating

Context: dft -scan, dft -test_points

Mode: setup

Usage

set_tristate_gating {**OFf** | **ON** | **Busdrivers** | **Scan** | *primary_input_or_output***...**}
[-Control {SEn | TEn}] [-Force_gating]

Description

Specifies how tri-state devices are controlled during scan chain shifting.

The set_tristate_gating command ensures tri-state devices are controlled to either prevent bus contention or be turned on during testing. If the necessary test logic already exists, none is inserted.

When enabled, test logic is inserted that controls tri-state devices as follows:

- **Multiple tri-state gates driving nets (bus net)** — One gate is turned on and the rest are turned off during testing.
- **Single tri-state gates driving nets (primary net or an internal net)** — All single tri-state gates are turned on for testing.

You can also specify which signal (SEN or TEN) controls the enable lines of tri-state devices.

By default, when the enable signal of a tri-state device is directly controlled by a primary input, by TIE0, or by TIE1, no gating is necessary and a force statement for the primary input is added to the load_unload procedure in the new procedure file. This behavior can be overridden by using the -Force_gating switch.

Arguments

- {**OFf** | **ON** | **Busdrivers** | **Scan** | ***primary_input_or_output...***}

  Required literal or repeatable string that specifies which tri-state devices to control during scan shifting. The default setting is off. Options include:

  OFf — no test logic is inserted to control tri-state devices. Default setting.

  ON — test logic is inserted when necessary to control all tri-state devices.

  Busdrivers — test logic is inserted to control tri-state devices driving bus nets.

  Scan — test logic is inserted to control tri-state devices used as scan inputs/outputs.

  primary_input_or_output — specifies a primary input or output pin. Test logic is inserted to control the tri-state devices driving the specified primary output pin(s) or driven by the specified primary input pin(s).

- -Control SEn | TEn

  An optional switch and literal pair that specifies the enable signal used to control tri-state devices. Literal options include:

SEn — Specifies the scan_enable signal. Default setting.

TEn — Specifies the test_enable signal.

- -Force_gating

  An optional switch that adds test logic to the enable lines of tri-states devices when these lines are directly controlled by primary inputs, or by TIE1, or by TIE0. When the enable line is directly controlled by a primary input, the tool adds the force statement for this primary input to the load_unload procedure in the procedure file.

Example 1

The following example uses the set_tristate_gating command to insert test logic and make all tri-state devices driving bus nets controllable via the SEN signal; it then reports the gated tri-state devices.

```
add_clocks 0 clk
set_tristate_gating on
set_bidi_gating scan
add_scan_pins c1 bidi_in1/X blkB1/blkA/utri2/A -top io out1
set_system_mode analysis
report_dft_check -tristate
```

```
-----------------------------------------------------------------------
                        Bidi                Primitive   Control  Control
Tri-state         State Direction  Gating ID          Signal   Driver
-----------------------------------------------------------------------
/bidi_in1         OFF   IN          YES   20          SEN      /or2/Y
/blkB1/blkA/utri3 OFF   --          YES   15          SEN      /udff0/Q
/blkB1/blkA/utri2 ON    --          YES   17          SEN      /or2/Y
/blkB1/blkA/utri1 OFF   --          YES   16          SEN      /or2/Y
-----------------------------------------------------------------------
```

Example 2

The following example uses the force_gating switch to insert test logic controlled by the SEN control signal on the enable line of /tpin_2. /tpin_2 is a tristate device driving primary output out4; its enable signal is directly controlled by the primary input /io_control1.

```
set_tristate_gating out4 -force_gating
report_dft_check -tri
```

```
-----------------------------------------------------------------
            Bidi                        Primitive   Control  Control
Tri-state   State Direction Gating      ID          Signal   Driver
-----------------------------------------------------------------
/tpin_1     ON     --         NO        44          SEN      /io_control
/tpin_2     ON     --         YES       43          SEN      /io_control1
/bidi_1     ON    OUT         NO        45          SEN      /io_control
/bidi_2     ON    OUT         NO        46          SEN      /io_control
/bidi_3     ON    OUT         NO        41          SEN      /io_control
-----------------------------------------------------------------
```

Related Commands

report_dft_check
report_test_logic
report_control_signals
set_bidi_gating
set_test_logic

set_tsdb_output_directory

Context: dft, patterns, unspecified

Mode: setup, analysis, insertion

Usage

set_tsdb_output_directory ***directory_path***

Description

Sets the tsdb output directory used by the process_dft_specification command.

The TSDB output directory defaults to *./tsdb_outdir* until the set_tsdb_output_directory command has been invoked. When either the get_tsdb_output_directory or set_tsdb_output_directory command is invoked, the tool verifies that the specified path is a writable directory if it already exists, or that the parent directory is a writable directory if it is not.

The process_dft_specification command is currently the only command that makes use of the tsdb directory.

Arguments

- ***directory_path***

  A required string value used to specify the TSDB output directory which is populated by commands such as process_dft_specification. The specified directory_path can be a relative path with respect to the current working directory or an absolute path. In either case, the directory must be a writable directory when it exists, or the effective parent directory must be a writable directory when it is to be created. The set_tsdb_output_directory command does not create the directory; it is created the first time a command needs to store information into it such as when the process_dft_specification command is invoked.

Example 1

The following example set the TSDB output directory to “./dir1”.

set_tsdb_output_directory dir1

Example 2

The following example shows a possible error than can be generated when setting the TSDB output directory.

set_tsdb_output_directory dir2/dir1

```
//  Error: The parent directory of the specified tsdb_output_directory
'/home/user1/dir2' does not exist.
```

Related Commands

process_dft_specification

get_tsdb_output_directory

set_visualizer_logging

Context: unspecified, all contexts

Mode: setup, analysis

Usage

set_visualizer_logging {{**-STArt** ***enhanced_dofile*** [-Replace | -Append] } | **-STOp}**

Description

Writes the commands entered into the Transcript window to the file specified by the *enhanced_dofile* argument.

Because only executable commands are written to *enhanced_dofile*, you can use the file like a normal dofile to recreate a previous view in DFTVisualizer.

Arguments

- **-Start**

  A required argument that specifies to begin writing each command that is entered into the Transcript window to the *enhanced_dofile* file.

- ***enhanced_dofile***

  A required string that specifies the name of the file into which you want to write the commands entered into the Transcript window. Only executable commands are written to this file.

- -Replace

  An optional switch that specifies to replace the contents of *enhanced_dofile* if that file already exists.

- -Append

  An optional argument that specifies to add the output of the logging operation to the *enhanced_dofile* file.

- **-Stop**

  A required argument that specifies to stop writing commands to the *enhanced_dofile* file.

Example

The following example unselects an object identified by its instance name:

```
set_visualizer_logging -start my_logfile
```

Related Topics

open_visualizer

set_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Usage

set_visualizer_preferences [-Compact | -NOCompact] [-NOQuery | -Query *threshold_number*] [-Split | -NOSplit] [-Display {DEBug | DESign | ALl}]

Description

Controls a subset of DFTVisualizer preferences for the Debug and Design windows.

This command controls the following three aspects of DFTVisualizer schematic display behavior:

- Display of gates
- Whether the tool will query you for confirmation before beginning to process data to display a large schematic
- Display of schematics in multiple sheets

There are invocation defaults for all the settings. When you use this command to change any of the settings, the specified preferences (along with any window displays affected by the settings) are updated immediately.

Arguments

- -Compact

  A switch that specifies to display only gates that could have a logical impact on the output results. This is the invocation default behavior.

  This switch is supported in the Debug window only.

- -NOCompact

  A switch that specifies to display all netlist gates, including buffers, inverters, ZVAL, and single-input bus gates. Many of these types of gates may not affect the logical output results, yet tend to clutter the display. The clutter is most noticeable when the gate level is set to primitive (set_gate_level Primitive).

  This switch is supported in the Debug window only.

- -NOQuery

  A switch that specifies for DFTVisualizer to display any display netlist you request, regardless of its gate count. This is the invocation default behavior for the Debug window.

- -Query *threshold*

  A switch and integer pair that specifies the maximum number of gates the Debug or Design window may display without first asking if you want to continue. Displaying netlist segments containing a relatively large number of gates can reduce performance. This switch

enables you to set a gate count threshold at which the tool will prompt you for confirmation that you do want to display a netlist segment that contains more than the *threshold* number of gates.

When determining whether to query you, the tool considers all gates, not just those that could have a logical impact on the output. The default threshold is 500 gates.

- -Split

  A switch that specifies to split the schematic into multiple schematic sheets and display just one sheet at a time. This is the invocation default behavior.

  In -Split mode, an off-sheet connector on a net indicates a connection to a gate on another sheet. Double click on the off-sheet connector to change the display to the other sheet. At the bottom of the display, to the right of the message area, the tool lists the sheet currently being displayed and the total number of sheets. Entering a sheet number in the sheet number entry box is another way to change the display to another sheet.

  This switch is supported in the Debug window only.

- -NOSplit

  A switch that specifies for DFTVisualizer not to split the schematic into multiple schematic sheets.

  This switch is supported in the Debug window only.

- -Display {DEBug | DESign | ALl}

  A switch and literal that specify the window(s) to which this command's settings apply. When you use the -Display switch, you have the following three choices:

  DEBug
  : A literal that applies the settings to the Debug window only. This is the invocation default behavior.

  DESign
  : A literal that applies the settings to the Design window only.

  ALl
  : A literal that applies the settings to both the Debug and Design window.

Examples

The following example changes the setting of the query threshold for both the Debug and Design window. The compaction and sheet splitting options continue to use the default settings.

```
set_visualizer_preferences -query 200 -display all
```

set_wrapper_chains

Context: dft -scan

Mode: setup, analysis

Prerequisites: At-speed test cells must be specified with the add_cell_models command or cell_type model attribute.

Usage

set_wrapper_chains [-EXClude *pin_names*…]
[{-INPUT_NUMber *integer*} | {-INPUT_MAX_length *integer*}]
[{-OUTPUT_NUMber *integer*} | {-OUTPUT_MAX_length *integer*}]
[-INPUT_Flops_reached {*integer* | Unlimited}]
[-OUTPUT_Flops_reached {*integer* | Unlimited}]
[-INPUT_Gate_levels {*integer* | Unlimited}]
[-OUTPUT_Gate_levels {*integer* | Unlimited}]
[{-Allow_internal_feedback | -No_internal_feedback | -Test_points}]
[-IO_registration {ON | OFf}]
[-AT_SPEED_TEST_FLOP_Insertion {ON | OFf}]
[-AT_SPEED_TEST_FLOP_Model ***modelname***]
[-Wrapper_extest_ports {ON | OFF}]
[-INClude_sub_chains {ON | OFf}]

Description

Specifies wrapper chains for scan insertion.

This command specifies the wrapper chains as follows:

- add_input_constraints CX
- set_output_masks On

Note

In the 2009_2 release and all subsequent releases, the set_wrapper_chains command performs both wrapper chain and core chain insertion in a single run. This is a change in behavior from previous releases and is not backward-compatible.

Information specified with this command is used during the wrapper cells analysis triggered with the analyze_wrapper_cells command. The wrapper cells analysis will identify the Input and Output wrapper cells, which will be stitched into the separate Input and Output wrapper chains. Wrapper cells are existing sequential elements accessible from primary inputs and primary outputs either directly or only via combinational logic. The Input wrapper chains are controlled by the SEN_IN type scan enable signals, while the Output wrapper chains are controlled by the SEN_OUT type scan enable signals. The Input and Output wrapper chains are used to provide proper test coverage of hierarchical designs during the INTEST and EXTEST modes:

- In INTEST mode, all inputs to submodules are controllable using the Input wrapper scan chains and all outputs are observable through the Output wrapper scan chains. This provides ability for the independent generation of a complete set of ATPG patterns for submodules.
- In EXTEST mode, all outputs from submodules are controllable using the Output wrapper scan chains and all inputs are observable through the Input wrapper scan chains. This provides ability to test/optimize the logic surrounding the submodules at the top-level without requiring internal visibility into the submodules (i.e. no need for core scan chains).

Note

You can specify either a number chain specification or a maximum length chain specification in a single session among the insert_test_logic, add_scan_partition, and set_wrapper_chains commands, but, you cannot specify both. For example, if you specify the number of chains for input wrapper chains using the set_wrapper_chains -input_number option, you cannot use the insert_test_logic -max_length option for the remainder of the scan cells.

Note

If non-scan elements are encountered during wrapper cell analysis, the set_wrapper_chains command automatically traces through the non-scan flip-flops until all available shared wrapper cells related to that particular branch are identified or other general terminating criteria, such as exceeding the limits or not finding any shared wrapper cells, have occurred.

You can modify the scan enable signals with the set_scan_enable_sharing command and the set_scan_enable command.

If you specify to insert both an at-speed test flip-flop at the beginning of each wrapper chain (-AT_SPEED_TEST_FLOP_Insertion option) and a lockup cell at the end of each wrapper chain (set_lockup_cell command's argument (-CApture_edge_at_scan_chain_input option), the order of stitching is: scan input, lockup cell, at-speed test flip-flop, and first wrapper chain cell. Inserted at-speed flip-flops are reported as part of the wrapper chain by the report_scan_cells command.

You can specify how wrapper cells are identified with the -No_internal_feedback, -Allow_internal_feedback, and -Test_points switches. By default, the tool identifies the wrapper cells by structurally tracing forward from the primary inputs until the first level of memory cells is reached and then, traces backward from the primary outputs until the last level of memory cells is reached. Figure 7-39 illustrates the default tracing where the traced logic and identified wrapper cells are shown in bold.

Figure 7-39. I/O Identification Default Tracing Mode

To control all the inputs of the combinational logic traced from the primary inputs, use the -Allow_internal_feedback switch. Figure 7-40 shows the gate inputs with feedback connections marked as *a* and *b*. Tracing backward from these inputs identifies one additional cell from the second level of memory cells to include in the output wrapper chains.

Figure 7-40. All Input Logic Tracing

Depending on the circuit topology, tracing such internal feedback may include an impractical number of core cells (not first or last level memory cells). In that case, you can control the feedback inputs on the logic gates by means of control points. In the above circuit, only the gate input *b* requires a control point because gate input *a* is controlled from its driver gate which is an identified input wrapper cell. In a multiple-phase testing of wrapper cells and core cells (i.e. hierarchical at-speed testing at a higher level of the design), the gate output marked as *c* cannot be observed when the testing of core cells is active and the testing of input wrapper cells is inactive. In that case, an observe point may be necessary at the gate output *c*.

The automatic insertion of such control and observe points can be specified using the -Test_points switch. Upon issuing the analyze_wrapper_cells command, the test point locations are identified along with the wrapper cell candidates. The identified test points are scheduled for insertion automatically. The actual insertion of the test points occurs later when the insert_test_logic command is issued.

Figure 7-41 highlights the logic that the tool adds as the control and observe points. The tool uses SEN_OUT type scan enable for the control points and SEN_IN type scan enable for the observe points, where possible.

Figure 7-41. Control and Observe Point Insertion

Arguments

- -Exclude *pin_names*

  An optional switch and a repeatable string that specifies the primary input/output pins to exclude from the wrapper cell identification process. The system clock pins (set, reset, clock, etc.) and test-related pins such as scan I/O, scan enable and test enable pins are excluded from the identification process automatically.

- -INPUT_NUMber *integer*

  An optional switch and an integer pair that specifies the number of scan chains for input wrapper cells. The default value for the number of input wrapper chains is 1.

Note

The number of scan chains specified with insert_test_logic -*number* applies to the entire design. The number of scan chains specified with other commands such as set_wrapper_chains -*input_number* and add_scan_partition -*number* is subtracted from this number.

- -INPUT_MAX_length *integer*

  An optional switch and an integer pair that specifies the maximum length of scan chains for input wrapper cells. The default value for the maximum length of the input wrapper chains is unlimited. The value of the set_wrapper_chains -input_max_length argument overrides the value of the insert_test_logic -max_length argument.

- -OUTPUT_NUMber *integer*

  An optional switch and an integer pair that specifies the number of scan chains for output wrapper cells. The default value for the number of output wrapper chains is 1.

Note

The number of scan chains specified with insert_test_logic -number applies to the entire design. The number of scan chains specified with other commands such as set_wrapper_chains -output_number and add_scan_partition -number is subtracted from this number.

- -OUTPUT_MAX_length *integer*

  An optional switch and an integer pair that specifies the maximum length of scan chains for output wrapper cells. The default value for the maximum length of the output wrapper chains is unlimited. The value of the set_wrapper_chains -*output_max_length* argument overrides the value of the insert_test_logic -*max_length* argument.

- -INPUT_Flops_reached {*integer* | Unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of sequential elements allowed to be reached during the forward tracing from a primary input. Default is 256.

- -OUTPUT_Flops_reached {*integer* | Unlimited}

  An optional switch and integer or literal pair that specifies the maximum number of sequential elements to be reached during the backward tracing from a primary output. Default is 256.

- -INPUT_Gates_level *integer*

  An optional switch and integer pair that specifies the maximum number of levels of combinational gates to be traversed during the forward tracing from a primary input until the first sequential element is reached. Default is 32.

- -OUTPUT_Gates_level *integer*

  An optional switch and integer pair that specifies the maximum number of levels of combinational gates to be traversed during the backward tracing from a primary output until the first sequential element is reached. Default is 32.

- {-Allow_internal_feedback | -No_internal_feedback | -Test_points}

  An optional switch that specifies the identification mode for input wrapper cells. Options include:

  -Allow_internal_feedback — Identifies additional cells with outputs fed back into the combinational logic between the primary inputs and the first level of registration cells. This is the default mode.

  -No_internal_feedback — Identifies the first level of registration cells by forward tracing from the primary inputs.

  -Test_points switch — Inserts control and observe points at the feedback connections instead of identifying additional cells from these feedback connections.

- -IO_registration {ON | OFf}

  An optional switch and literal pair that specifies whether to automatically use the current I/O registration mechanism to add wrapper cells for the primary inputs and outputs that fail wrapper cell identification. Wrapper cell identification will fail in the following cases:

  - If tracing limits are exceeded during wrapper cell identification for an I/O.
  - If the identified sequential cell is part of an existing scan chain.
  - If a blackbox is encountered during the wrapper cell identification for an I/O.
  - If no sequential cells were identified during wrapper cell identification for an I/O.
  - If the currently identified sequential cell is already identified as a wrapper cell for an I/O of the opposite direction.

  Options include:

  ON — A new wrapper cell is inserted for each failed primary input and output. This is the default.

  OFF — A wrapper cell is not automatically inserted for each failed primary input and output.

- -AT_SPEED_TEST_FLOP_Insertion {ON | OFf}

  An optional switch and literal pair that specifies whether to automatically insert an at-speed test flip-flop at the beginning of each wrapper chain. Default is On. If a preferred model is not specified with the -AT_SPEED_TEST_FLOP_Model switch, the tool uses the first DFF model declared by the add_cell_models command;

- -AT_SPEED_TEST_FLOP_Model ***modelname***

  An optional switch and literal pair that specifies the DFF model to use for an at-speed test flip-flop when more than one DFF model is declared via the add_cell_models command.

- -Wrapper_extest_ports {ON | OFF}

  An optional switch and literal pair that specifies to insert direct access extest scan I/O ports for the wrapper chains inserted during the block-level insertion phase of an SoC DFT flow. The default is ON.

  In order to have the extest scan input port at the block-level boundary, for each wrapper chain, the tool inserts a multiplexer in front of the scan input pin of the first scan cell in the chain. The 0-input of the multiplexer is connected to the conventional scan input port at the module boundary. The 1-input of the multiplexer is connected to the new extest scan input port at the module boundary. The default naming of these new ports can be changed using the -Wrapper_intest and -Wrapper_extest switches of the set_scan_pins command. The enable pin of the multiplexer is also connected to the module boundary with a new port named “wrapper_extest”. The default naming of this port can be changed using the -WEN switch of the set_scan_insertion command.

  In order to have the extest scan output ports at the block-level boundary, for each wrapper chain, the tool inserts a buffer after the last scan cell in the chain. The output pin of the last

scan cell feeds the buffer input and the conventional scan output port at the module boundary. The output of the buffer feeds the new extest scan output port at the module boundary. If a lockup cell is to be inserted after the last scan cell in the chain, it is inserted to feed the buffer input and the conventional scan output port.

When this switch is used and *-channels* is specified with the set_edt_options command, the subchain dofile generated using the write_subchain_setup command includes a second group of subchain declarations for the wrapper chains accessed through the new extest ports.

- -INClude_sub_chains {ON | OFF}

  An optional switch and literal pair that specifies to stitch defined subchains into wrapper chains if the following conditions are true:

  - The subchain's scan path is traceable and all cells on the scan path can be identified.
  - For a module-based subchain, all scan cells on the subchain's scan path are reachable from primary I/Os.
  - For a library model-based subchain, all scan cells on the subchain's scan path are either reachable from primary I/Os or their data and non-scanout output ports are not connected externally.

  Options include:

  ON — Qualified subchains (those that meet the listed conditions) are stitched into wrapper chains. This is the default.

  OFF — The subchain is not stitched into a wrapper chain.

Example

The following example excludes the primary I/O pins *a* and *b* from wrapper cell identification; sets up four input wrapper scan chains and eight output wrapper scan chains; sets sen1 and sen2 for the scan enable pin names of the input and output wrapper chains and reports the sequential cells identified per I/O pin.

```
add_cell_model LATX -type dlat G D
add_cell_model INVX -type inv
set_wrapper_chains -input_num 4 -output_num 8 -exclude a b
set_scan_enable sen1 -wrapper_chain -input
set_scan_enable sen2 -wrapper_chain -output
set_system_mode analysis
analyze_wrapper_cells
report_wrapper_cells
insert_test_logic -clock merge -edge merge
```

Related Commands

add_tied_signals
report_environment
report_wrapper_cells
set_lockup_cell
set_scan_enable
set_scan_enable_sharing
set_output_masks
set_registered_io
set_scan_insertion
set_scan_pins
write_design
write_subchain_setup

set_xbounding_options

Context: dft -scan

Mode: setup

Usage

set_xbounding_options [-xbounding_enable ***xbounding_enable_name***]
[-connect_to {**existing_scan_cell** | **new_scan_cell** | **constant_value**}]
[-model ***name***] [-scan_drive_limit ***integer***]
[-mcp_bounding_enable ***mcp_enable_name***]
[-exclude_sdc_cross_domain_paths {**off** | **on**}]
[-test-clock {**off** | **on**}]

Description

Sets up the parameters for X bounding insertion. The analysis is triggered by the analyze_xbounding command.

You use this command to set various options that control how the tool performs X-bounding analysis. These options include:

- The name of the enable signals that activate the bounding logic.
- The type of bounding (new scan cells, existing scan cells, or constant values) that is used to block the X source.
- Activate (or disable) bounding of cross-clock domain paths as specified by the SDC file.

The actual analysis must be triggered by issuing the analyze_xbounding command.

Arguments

- -xbounding_enable ***xbounding_enable_name***

  An optional switch and literal pair used to specify the name of the of the X-bounding enable signal.

- -connect_to {**existing_scan_cell** | **new_scan_cell** | **constant_value**}

  An optional switch and literal pair used to specify whether the X sources are controlled by existing or new scan cells or whether the X source is blocked with a constant value.

  Choose from one of the following:

  existing_scan_cell — Specifies using existing scan cells to control the X sources.

  new_scan_cell — Specifies using new scan cells to control the X sources. When using new scan cells for X bounding, the optional -model switch can be used to specify the scan cell library cell model. When this switch is not used, the scan cell model specified either by the add_cell_model command or the cell_type ATPG library attribute is used.

  constant_value — Specifies using either an AND gate to block the X source with a constant 0 or an OR gate to block the X source with a constant 1. The specific

bounding value (0 or 1) is determined by performing a simple analysis of the logic that will be forced to the constant value.

- -model ***name***

  An optional switch and string pair used in conjunction with the new_scan_cell switch and specifies the scan cell library model.

- -scan_drive_limit ***integer***

  An optional switch and integer pair that specifies a limit on how many Mux-type control points one scan cell can drive for bounding. Having one scan cell drive a large number of Mux control points can reduce test coverage, due to structural dependency. 0 means there is no limit. The invocation default is 4.

- -mcp_bounding_enable ***mcp_enable_name***

  An optional switch and string pair that specifies the name of an alternate enable signal for bounding at-speed paths. This provides independent control of these bounding muxes as the tool must bound multicycle paths during at-speed testing.

- -exclude_sdc_cross_domain_paths {**off** | **on**}

  An optional switch and literal that specifies whether the tool bounds cross clock-domain paths that are specified in the SDC file.

  A cross clock-domain path occurs when memory elements driven by one clock domain, feed logic that is observed (captured) by memory elements that are operated by a different clock.

  Cross clock-domain paths are typically specified as false paths in the SDC file. In that case, these paths are identified as potential X sources and bounding muxes are inserted near the capture point.

  In some cases, this might constrain the clock sequences that are applied during Logic BIST such that these paths will never be sensitized. However, if your hardware is designed such that certain cross clock_domain interactions are not tested during Logic BIST, and you choose to not test these the cross clock_domain paths by constraining the capture sequences, then there is no need to insert bounding muxes.

  The -exclude_sdc_cross_domain_paths switch instructs the tool to ignore false paths that are specified in the SDC file during analysis.

Note

In order to guarantee that all cross clock domains are targeted for x-bounding, you must specify all cross clock domain paths as false paths in the SDC file. This can be done indirectly by using the set_clock_groups command to assign each clock to a separate group.

- -test_clock{**off** | **on**}

  An optional switch and literal pair that determines the clock signal for the newly inserted scan cells that will drive the test_mode input of the x-bounding muxes. This option must be used in conjunction with the -connect_to new_scan_cell option.

  off — A literal that specifies that the clock signal for the newly created scan cells will be chosen to match the clock domain of the flops in the fanout cone of the x-bounding mux.

  on — A literal that specifies that the newly created scan cells will be driven by the test clock signal as defined with the"set_scan_insertion -tclk" command.

Example

The following command specifies non-default enable names for the primary enable signal as well as the false and multicycle path enable prior to performing the x-bounding analysis:

```
set_xbounding_options -xbounding_enable lbist_enable \
        -mcp_bounding_enable mcp_path_enable

analyze_xbounding
```

Related Commands

analyze_xbounding

insert_test_logic

report_xbounding

set_xclock_handling

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

Mode: setup, analysis

Usage

set_xclock_handling {**Retain** | **X** [-Pessimistic_simulation {ON | OFf}]}

Description

Specifies whether the sequential element model outputs X when any of its clock inputs become X.

By default, the flip-flop and latch default models are used for sequential elements. These default models retain their output values when their clock value becomes X so long as the other input values do not cause the stored output value to change, regardless of whether the clock was at a 0 or 1. The Set Xclock Handling command lets you override this behavior during Tessent FastScan or Tessent TestKompress simulation, causing the stored output value to become X.

This command has no effect during DRC simulation. The default flip-flop and latch behavior is used during DRC simulation.

Arguments

- **Retain**

  A required literal that specifies to use the default sequential element model. This is the default.

- **X** [-Pessimistic_simulation {ON | OFf}]

  A required literal and optional switch set that changes the sequential element model so that during simulation the output of the element is set to X when any of its clock inputs become X.

  -Pessimistic_simulation OFf

  Default. An edge-trigger flip-flop capture value is set to X only when X is observed at the clock port of the effective clock edge. For a leading edge flip-flop, the effective clock edge is leading edge, so the flip-flop is masked out only when X is present at the clock port during the leading edge event. If X happens at the trailing edge, then the leading edge flip-flop value is not affected.

  -Pessimistic_simulation ON

  An edge-trigger flip-flop capture value is set to X whenever X is observed at the clock port, regardless of the effective edge of the clock port. This option may be required for some Verilog simulators to prevent a simulation mismatch.

Example 1

The following example sets the outputs of sequential elements to X only when X is observed at the clock inputs at the effective clock edge during pattern simulation.

```
set_xclock_handling x
```

Example 2

The following example sets the outputs of sequential elements to X when any of their clock inputs become X during pattern simulation.

```
set_xclock_handling x -pessimistic_simulation on
```

set_z_handling

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

Mode: setup

Usage

set_z_handling {**Internal** ***state***} | {**External** ***state***}

Description

Specifies the simulation handling for high impedance signals on internal and external tri-state nets.

The set_z_handling command specifies how to handle the high impedance state for internal and external tri-state nets during pattern simulation. If the high impedance value can be made to behave as a binary value, certain faults may become detectable. If you do not use this command to set the Z handling, the default value is X.

This command does not impact expected values on tri-state and bidirectional pins in the final saved patterns. The command changes the behavior of how Z values are simulated on internal nets. By default, Z values are treated as X values when fed into another internal gate, such as an AND gate. However, a Z value on a tri-state output pin is not converted to an X by default.

Note

This command applies to simulation only; it does not replace Z values in generated patterns. To alter the Z values in patterns to match the Z handling specified with this command, use the -Noz switch with the write_patterns command when you save the patterns.

Arguments

- **Internal** ***state***

  A literal pair that specifies how the tool simulates high impedance values for internal tri-state nets. The literal choices for the *state* option are as follows:

  X — A literal that specifies to treat high impedance states as an unknown state. This is the default behavior upon invocation of the tool.

  0 — A literal that specifies to treat high impedance states as a 0 state.

  1 — A literal that specifies to treat high impedance states as a 1 state.

- **External** ***state***

  A literal pair that specifies how the tool simulates high impedance values for external tri-state nets. The literal choices for the *state* option are as follows:

  X — A literal that specifies not to measure high impedance states; they cannot be distinguished from a 0 or 1 state. This is the default behavior upon invocation of the tool.

0 — A literal that specifies to treat high impedance states as a 0 state that can be distinguished from a 1 state.

1 — A literal that specifies to treat high impedance states as a 1 state that can be distinguished from a 0 state.

Z — A literal that specifies to uniquely measure the high impedance states; they can be distinguished from both a 0 or 1 state.

Example

The following example specifies to treat high impedance values as 1 states when they feed into logic gates, and as 0 states at the output of the circuit, then performs a pattern simulation run:

```
set_z_handling internal 1
set_z_handling external 0
set_system_mode analysis
create_patterns
```

Related Commands

write_patterns

set_zhold_behavior

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

Mode: setup, analysis

Usage

set_zhold_behavior **OFf** | **ON**

Description

Specifies whether ZHOLD gates retain their state values.

The ZHOLD gate allows the tool to model tri-state nets so that they can retain the previous state value when the net goes to a high impedance (Z) value. ZHOLD gates normally require clock sequential patterns to utilize this capability. But, if a ZHOLD gate is set to a fixed binary value when the clocks are off, the ZHOLD gate can retain that value for combinational patterns. The combinational fault simulation does not consider the fault effect on the retained value.

By invocation default, the tool assumes the retained value is always the clock off_state value for both the good machine and the faulty machine simulation.

The tool cannot assume a state from a previous scan pattern or from a load operation. The amount of time the ZHOLD gate can hold its value is limited to the number of sequential clock cycles.

Arguments

- **OFf**

  A literal that specifies not to allow ZHOLD gates to retain values.

- **ON**

  A literal that specifies for the tool to use ZHOLD gates to retain values subject to restrictions that the rules checker identified. This is the default behavior upon invocation of the tool.

Example

The following example specifies that the ZHOLD gates are not allowed to retain their previous values:

```
set_zhold_behavior off
```

Related Commands

report_gates

set_learn_report

setenv

Context: unspecified, all contexts

Mode: all modes

Usage

setenv ***variable_name*** [*value*]

Description

Sets a shell environment variable within the tool environment.

The setenv command sets or changes the value of a shell environment variable within the current tool environment, enabling the tool to access the variable for the rest of the session just as if the variable had been set from the shell. This command is useful if you ever find that the tool requires an environment variable that was not set prior to invocation.

Note

A shell variable you set with this command is set only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To unset the variable before you exit, use the unsetenv command. To view the value of the variable, use the command to pass an appropriate shell command (**echo** or **env** for example) to the operating system for execution.

Arguments

- ***variable_name***

  A required string that specifies the name of the variable.

- *value*

  An optional string that specifies the value of the variable.

Example

The following example assigns a value to the environment variable SSH_AGENT_PID, then displays its value:

```
setenv SSH_AGENT_PID  66216
system echo $SSH_AGENT_PID
```

```
66216
```

Related Commands

unsetenv

shutdown_sid_tester

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

Mode: setup, analysis

Usage

shutdown_sid_tester

Description

Terminates the currently active SID tester process that was created by executing the launch_sid_tester command.

Example

The following example shuts down the SID tester process.

```
shutdown_sid_tester
```

Refer to "Executing an IJTAG Test Pattern" in *Tessent SiliconInsight User's Manual for Tessent Shell* for a sample usage scenario.

Related Topics

launch_sid_tester

simulate_clock_pulses

Context: all contexts

Mode: setup, analysis

Prerequisites: The flat model must already exist, and you must have previously used the add_clocks command to define specified gate_pin objects as clocks.

Usage

simulate_clock_pulses ***obj_spec*** [-repetitions *number*]

Description

Pulses one or more clocks within the current simulation context.

The simulate_clock_pulses command pulses the specified clock(s) within the current simulation context. Before issuing this command, you must previously use the add_clocks command to define the specified gate_pin objects as clocks.

The default number of clock pulses is 1 unless you specify a different number with the -repetitions switch. If any forces have been queued with previous add_simulation_force commands, the tool automatically simulates these forces before pulsing the clocks.

Arguments

- ***obj_spec***

  A required value that specifies one or more gate_pin objects.

- -repetitions *number*

  An optional switch and integer pair that specifies the number of clock pulses to simulate. If you don't use this switch, the default number of clock pulses is 1.

Example

The following example shows the use of the simulate_clock_pulses command:

```
SETUP> add_clocks myInst1/OUT -internal
SETUP> add_clocks myInst2/OUT -internal -pin_name myClk
SETUP> simulate_clock_pulses [get_gate_pins {myInst1/OUT myClk} ] -repetitions 2
```

Related Commands

add_simulation_force
delete_simulation_forces
get_simulation_value_list
report_simulation_forces
simulate_forces

simulate_forces

Context: all contexts

Mode: setup, analysis

Prerequisite: The flat model must already exist

Usage

simulate_forces

Description

Simulates the queued forces in the current simulation context.

The simulate_forces command simulates all the queued simulation forces previously added with the add_simulation_force command. This command operates only within the current simulation context.

Arguments

None.

Example

The following example shows the effect of adding a force on a particular gate_pin object, and then issuing the simulate_forces command to trigger the tool to simulate that force:

ANALYSIS> **set_current_simulation_context myContext**
ANALYSIS> **add_simulation_force [get_gate_pins I1] -value 1**
ANALYSIS> **report_simulation_forces**

```
Value   Object
------  ------
1       I1
```

ANALYSIS> **get_simulation_value_list [get_gate_pins I1]**
{ X }

ANALYSIS> **simulate_forces**
ANALYSIS> **get_simulation_value_list [get_gate_pins I1]**
{ 1 }

Related Commands

add_simulation_force
delete_simulation_forces
get_simulation_value_list
report_simulation_forces
simulate_clock_pulses

simulate_patterns

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Usage

simulate_patterns [-source {external | bist | random}]
[-store_patterns {all | effective | none}]

Description

Performs simulation by applying the specified pattern source.

The simulate_patterns command performs a good-machine simulation, and also performs a fault simulation only when there is a current fault list. By default, the command simulates the external patterns previously loaded with the read_patterns command, and the tool retains the pattern order (disables pattern classification) after simulation. The command optionally stores the patterns to the internal pattern set.

The command issues a warning before performing a good-machine simulation when no patterns are stored.

When executing the simulate_patterns command, the tool analyzes the design to determine if split capture cycle or clock_off simulation is needed. If so, the tool turns these functions on automatically. When the pattern source is set to BIST and an NCP (named capture procedure) is used, the tool checks the NCP to determine the necessary sequential depth and other environment settings (such as multi-load or hold_pi) before starting fault simulation. For more information, refer to “Example 2” on page 2010.

Arguments

- -source {external | bist | random}

  An optional switch and literal pair that specifies which pattern source to simulate. By default, the command simulates the external patterns previously loaded with the read_patterns command.

  The “bist” option specifies that the tool simulate only BIST patterns. Note that when running from BIST mode (LSFR is defined or in EDT_LBIST mode) and no external patterns are loaded, the default pattern source is BIST. There must be at least one defined LFSR. The existence of such an LFSR means that the design successfully passed the BIST rules checking when leaving Setup.

  You can use the -store_patterns switch to store the BIST patterns when performing good simulation; however, the “-store_patterns effective” option is not allowed. These patterns are different from normal patterns in that they specify the excess values that occur on short scan chains during the load and unload process. Also, the last pattern does not contain an unload of the scan chains.

  The “random” option specifies the random patterns, capture clock, and observation point specified previously. You can use this option to create patterns for random pattern testable

faults. For fault simulation, you can use this option to evaluate the expected random pattern test coverage.

- -store_patterns {all | effective | none}]

  An optional switch and literal pair that specifies if the simulated patterns should be added to the internal pattern set. By default, the command stores no patterns. However, you can optionally store all patterns or store only the patterns that were effective at detecting additional faults.

Example 1

The following example performs a good-machine simulation using the current BIST patterns and then adds those patterns to the internal pattern set:

simulate_patterns -source bist -store_patterns all

Example 2

The following example applies an NCP named “cap_depth3” and then initiates a BIST pattern fault simulation. Note that before running the simulation, the tool first analyzes the NCP and then automatically changes the sequential depth to 3 and also issues a “set_split_capture_cycle on” command.

> **add_bist_capture_range cap_depth3 0 511**

> **simulate_patterns -source bist**

```
//  Warning: Current sequential depth is not large enough to apply named capture procedures.
//           Automatically increase the sequential depth to be 3.
//  largest sequential test depth = 1, largest controllability depth = 1,
//  largest observability depth = 1
//  largest test depth is on /tck (57) (control = 0, observe = 1)
// | ---------------------------------------------------------------------------------
// |          Current split capture setting:      Off
// |                         DRC requirement:      Yes (C3)
// |                                 Calling:      set_split_capture_cycle on
// |
// |   Current clock off simulation setting:      Off (optimal)
// | ---------------------------------------------------------------------------------
//
//  ----------------------------------------------------------------------
//  Simulation performed for #gates = 9002  #faults = 25443
//  Run fault simulation without storing patterns.  pattern source = 512 BIST patterns
//  ----------------------------------------------------------------------
//  #patterns  test      #faults  #faults    # eff.    # test         process
//  simulated  coverage  in list  detected   patterns  patterns       CPU time
…
//   64         47.85%    12158     13285       64         0          0.50 sec
//   128        53.66%    10121      2037       56         0          0.56 sec
//   192        56.52%     9118      1003       38         0          0.60 sec
…
```

Related Commands

delete_patterns
read_patterns
write_patterns

sizeof_collection

Context: unspecified, all contexts

Mode: all modes

Usage

sizeof_collection ***collection***

Description

Returns the number of objects in a collection.

The sizeof_collection command is an efficient way to determine how many objects are in a collection.

Arguments

- ***collection***

  A required string that specifies the collection for which to get the number of objects. The returned size for an empty collection is 0.

Example 1

The following example shows a simple way to find out how many instances of cell module the current design has.

```
puts [sizeof_collection [get_instances -filter {type == “cell”}]
56781
```

Example 2

The following example shows the size of a collection as it gets appended to.

```
set a “”
puts [sizeof_collection $a]
0

append_to_collection a b[7]
{b[7]}

puts [sizeof_collection $a]
1

append_to_collection a {b c d}
{b[7] b c d}

puts [sizeof_collection $a]
4
```

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
index_collection
is_collection
range_collection
remove_from_collection
sort_collection

sort_collection

Context: unspecified, all contexts

Mode: all modes

Usage

sort_collection ***collection criteria*** [-descending]

Description

Sorts a collection based on the value of one or more attributes, resulting in a new sorted collection.

The sort is in ascending order by default. In an ascending sort, objects that have Boolean attributes set to false are first followed by Boolean objects that have attributes set to true. The sorting considers the default values of the attributes when the attribute has never been set on the object.

To sort the objects in reverse order, specify the -descending option.

Arguments

- ***collection***

  A required string that specifies the collection to be sorted.

- ***criteria***

  A required value that specifies an attribute name or a Tcl list of attribute names to use as sort keys. The order of the attribute name in the list controls the order of priority while sorting.

- -descending

  An optional switch that sorts the collection in reverse order. By default, the sort proceeds in ascending order.

Example

The following example illustrates the use of two attributes to sort a collection. Notice how the first attribute has higher priority than the second attribute in the sorting.

```
register_attribute -name att2 -value_type string -default a
register_attribute -name att1 -value_type string -default a

set_attribute_value {ba bb} -name att1 -value b
{ba bb}

set_attribute_value {ab bb} -name att2 -value b
{ab bb}

sort_collection [get_inst {aa ab ba bb}] {att1 att2}
{aa ab ba bb}

sort_collection [get_inst {aa ab ba bb}] {att2 att1}
{aa ba ab bb}
```

sort_collection [get_inst {aa ab ba bb}] {att2 att1} -descending {bb ab ba aa}

Related Commands

add_to_collection
append_to_collection
compare_collections
copy_collection
filter_collection
foreach_in_collection
index_collection
range_collection
is_collection
remove_from_collection
sizeof_collection

system

Context: unspecified, all contexts

Mode: all modes

Usage

system ***os_command***

Description

Passes the specified command to the operating system for execution.

The system command executes one operating system command without exiting the currently running application.

Arguments

- ***os_command***

  A required string that specifies any legal operating system command.

Example

The following example invokes a synthesis script to synthesize generated RTL, and reloads the synthesized view:

```
write_design -output_file rtl/my_module.v
system “cd synth_work; ./synth_script”
read_verilog gate/my_modul.v -force
set_current_design mytop
```

Example 2

The following example performs an ATPG run, then displays the current working directory without exiting the tool:

```
set_system_mode analysis
create_patterns
system pwd
```

trace_flat_model

Context: all contexts

Mode: setup, analysis

Prerequisites: This command can only be used after the design is flattened to the simulation model; this occurs when you first exit setup mode to enter analysis mode, or when you issue the create_flat_model command.

Usage

trace_flat_model **-from** ***from_obj_spec*** [-to *to_obj_spec*] **-direction** forward | backward
[-tag_condition *tag_attribute_expression*] [-stop_condition *stop_attribute_expression*]
[-controllability {controlling | constant | unblocked | connected}]
[-stop_at_first_tag {On | Off}] [-max_levels {unlimited | *number*}]
[-latch {normal | transparent}] [-map_tag_to_design_module_boundary { ON | OFf }]
[-store_polarity_to_tagged_objects *polarity_var_name*]
[-store_paths_to_tagged_objects *tag_path_var_name*]
[-store_path_to_stop_point *stop_path_var_name*]

Description

Traces the flat model taking into account the current simulation context.

For information about simulation contexts, refer to the set_current_simulation_context command description.

The tool begins tracing at the objects specified by the -from option. The direction is either forward or backward. You can use equations based on attributes found on gate_pin objects to specify the gate_pins that you want the tracing to stop on, or the gate_pin objects you want to tag when reached.

This command returns a collection of reached gate_pins for which the tag condition evaluates to true. When the -tag_condition argument is specified, this command returns a collection of all gate_pin objects the trace stopped on. You can also provide a Tcl variable in which to store the path between the start object and the tagged objects so that you can display them in DFTVisualizer as shown in two of the following examples.

Note that when using this command in setup mode, you may see the following error message because no simulation data is available:

```
Error: Controllability 'unblocked' is the default, but no simulation
data exists in the current simulation context ...
```

To avoid this error, you must specify the “-controllability connected” trace option.

Arguments

- **-from *from_obj_spec***

  A required option that specifies a single name of a gate_pin, pin or port object, a Tcl list of names of gate_pin, pin, or port objects, or a collection of gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

  Pins that are preserved in the flat model are either all pins on instances of type cell, or pins on instances of type design on which an attribute is set to a non-default value and the set_attribute_option -preserve_boundary_in_flat_model option of that attribute is set to true. These attributes must have been defined on the pin in setup mode prior to creating the flat model.

  When controllability is set to constant or controlling, you can only specify one object in the from_obj_spec. The tracing is performed starting from each object specified in the from_obj_spec argument.

- -to *to_obj_spec*

  An optional option that directs the trace operation to stop at the objects specified by to_obj_spec and to tag them, if reached. The to_obj_spec argument can be a single name of a gate_pin, pin or port object, a Tcl list of names of gate_pin, pin, or port objects, or a collection of gate_pin, pin, or port objects. When using pin names or objects, these pins must have been preserved in the flat model.

  You can use this switch to specify objects as stop points in addition to the objects specified with the -stop_condition switch, and to specify which objects will be tagged in addition to the objects specified with the -tag_condition switch.

- **-direction** forward | backward

  A required string that specifies whether tracing should be performed forward or backward from the defined start object(s).

- -tag_condition *tag_attribute_expression*

  An optional switch and string pair that specifies the tag condition for the trace. All of the reached design objects fulfilling the defined tag condition are returned. The default is an empty string; in this case, the trace stop points are returned. In a non-default case, the trace stop points are not returned if the tag condition expression is not true for these objects. The format for the *tag_attribute_expression* is identical to the -filter option used for the get_pins command. See section "Attribute Filtering Equation Syntax" for more details on filtering attribute equation formats.

- -stop_condition *stop_attribute_expression*

  An optional switch and string pair that defines the stop condition for the trace. The default is an empty string; in this case, the trace is stopped at the gate_pin failing the specified controllability requirements. The pins of memory or dff primitives are always stop points even when not explicitly specified.

- -controllability {controlling | constant | unblocked | connected}

  An optional switch and literal pair that defines the controllability requirements before traversing a gate.

  controlling— Gates are only traversed if the other inputs have simulated constant values making a single input directly controlling the output. Inversions are possible and information about them is stored in the variable specified by the -store_polarity_to_tagged_objects argument.

  constant — Only pins with a constant value are traversed. This mode is useful to find the source of a constant value.

  unblocked — Gates are only traversed if the other inputs do not have simulated constant values making the input blocked from the output pin. Only unblocked combinational paths are traversed.

  connected—With this value, gates are always traversed when reached. You should use this option when this command is used in setup mode because no simulation data exists yet.

- -stop_at_first_tag {On | Off}

  An optional switch and literal pair that specifies whether to interrupt tracing when the first tag object with the specified controllability is found. This is useful when checking for at least one occurrence of a given condition in the fanin or fanout of a pin without wasting time to find them all.

- max_levels {unlimited | *number*}

  An optional switch and value pair that defines the maximum level of gates to trace. The default is unlimited but you can specify a positive integer to limit the level count.

- -latch {normal | transparent}

  An optional switch and literal pair that specifies the behavior of tracing through a latch.

  normal — A latch is traced through only if the simulated value in stable_after_setup makes the latch satisfy the specified controllability requirements.

  transparent — All latches are traced through regardless of the value at the latch enable port.

- -map_tag_to_design_module_boundary { ON | OFf }

  An optional switch and literal pair that specifies whether to map the tag points to gate_pins outside of library cells. When this switch is ON, the command maps the tag points to gate_pins outside of library cells and never inside the cells. Library cells can be either cells from a ATPG library or `celldefine Verilog modules.

- -store_polarity_to_tagged_objects *polarity_var_name*

  An optional switch and string pair that specifies the name of a Tcl variable in which to store a list of 0 or 1 values to reflect the presence of an inversion between the start object and the tag object(s). If an inversion exists between the start and tag object(s), the value 1 is stored

in the list; otherwise, the value 0 is stored. This option can only be used with the -controllability controlling option; inversion is meaningless in all other cases.

- -store_paths_to_tagged_objects *tag_path_var_name*

  An optional switch and string pair that specifies to store a list of paths in the variable *tag_path_var_name*. Each path holds a collection containing all of the traversed gate_pins between the start object and the tagged object(s). The list is in the order specified by the -direction option and includes the start object and the reached tagged object.

- -store_path_to_stop_point *stop_path_var_name*

  An optional switch and string pair that specifies to store the path between the start object and the gate_pin where the trace stopped independent of whether the stop point is a tagged object. This switch is only valid when -direction is specified as backward and -controllability is specified as controlling; in this case, only one start object is supported and only one stop point can ever be found. This mode is useful to trace a clock source and display the path to the stop point if an appropriate source is not reached.

Example 1

The following example traces the unblocked fanout of a pin. Because no -tag_condition is specified, the tool automatically tags every node with no unblocked fanout. The paths to the tag elements are stored in the Tcl variable tag_paths. The foreach command iterates on the list of paths and the add_display_instance command displays the paths in the DFTVisualizer Debug window. Figure 7-42 shows the resulting schematic drawn in the debug window.

```
set start_pin [get_pins BOB_CKB2_1_MBIST1_MBIST_I1/CMP_EN_R_reg/Q]
trace_flat_model -from $start_pin \
   -direction forward \
   -store_paths_to_tagged_objects tag_paths

foreach path $tag_paths {
   add_display_instance $path
}
mark_display_instances $start_pin -marking_index 5
```

Figure 7-42. Debug Window After add_display_instance Command

Example 2

The following example performs a controlling walk forward from a clock source and collects any MU111 multiplexers traversed along the way. The paths from the start point to all tag elements are displayed in the Debug window as shown in Figure 7-43. Notice how the set_gate_report drc command is set to stable_after_setup; this corresponds to the simulation conditions used by the trace_flat_model command. You can see that the two muxes have a controlling value on the select pin. Multiplexers without a controlling value on the select pin were not traversed and therefore their output pins were not tagged.

```
trace_flat_model -from CLKB \
      -tag {module_name == MU111 && direction == output} \
      -direction forward \
      -store_paths tag_paths
      -controllability controlling
set_gate_report drc stable_after_setup
foreach path $tag_paths { add_display_inst $path }
```

Figure 7-43. Result of trace_flat_model While Tagging Multiplexers

Example 3

The following example uses the trace_flat_model command to find paths between output and inout pins of modb to pins of modd. Notice how the -stop_at_first_tag option is set to on, ensuring that only one path from each modb pin is reported.

```
set_system_mode setup
set_attribute_value [get_instances -of_module {modb modd}]
   -name preserve_boundary
set_system_mode analysis
trace_flat_model
   -from [get_pins u4/* -direction {output inout}]
   -direction forward
   -stop_condition preserve_boundary
   -tag_condition {module_name == modd}
   -stop_at_first_tag on
   -store_paths_to_tagged_objects paths

if {[llength $paths] > 0} {
   puts "pins of modb fanning out to modd:"
   foreach path $paths {
      puts [get_single_name [index_collection $path 0]]
   }
}
```

Related Commands

get_trace_flat_model_option
set_trace_flat_model_options
set_current_simulation_context

uncompress_layout

Context: patterns -scan_diagnosis

Mode: analysis

Usage

LDBs created after v2013.2: uncompress_layout [-REPlace]

LDBs created with v2013.2 and before: uncompress_layout ***layout_database_name*** [-REPlace]

Description

Uncompresses a Layout Database.

You must open the Layout Database using the open_layout command before performing the uncompress operation.

See "Layout Database Compression and Decompression" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- -REPlace

  An optional switch for overwriting an existing file.

- ***layout_database_name***

  A required string that specifies the location to and the name of the Layout Database to be created during the uncompress operation.

Related Commands

compress_layout
create_layout
open_layout

uniquify_instances

Context: all contexts

Mode: insertion

Usage

uniquify_instances ***obj_spec*** [-module_uniquification_suffix *module_uniquification_suffix*] [-silent]

Description

Uniquifies the module definition of the specified instance objects found in obj_spec.

If the instances specified by obj_spec have a module definition that is used more than once below the current design, the module name is made unique by appending the specified -module_uniquification_suffix value to its name.

This command returns a collection of all modules associated with the instances in the obj_spec irrespective of whether the module name was uniquified or not.

Arguments

- ***obj_spec***

  A required value that specifies the instances whose module name is to be uniquified. The obj_spec value must be a single instance name, a Tcl list of instance names, or a collection of instances as generated by the get_instances command

- -module_uniquification_suffix *module_uniquification_suffix*

  An optional switch and value pair that specify a string to append to the module name if the instance's module definition is used more than once below the current design. The default module_uniquification_suffix is "_#" where # begins at 1 and increments by one until a unique module name is found.

- -silent

  An optional argument that specifies to ignore specified instance objects that do not exist or instances that are not of type design, and to not generate error messages.

Example 1

The following example requests all instances found within u2 to be uniquified. All instances of design modules in this example are instances of module moda and modb. From the result of the command, you can see that there are only three instances of moda within the current design and they are all within the u2 instance. Once the first two instances were uniquified to moda_1 and moda_2, the third instance was not changed as it was now unique. Because mod_b was uniquified, this means that it is used in other instances of the current design.

uniquify_instances [get_instances u2/* -of_type design]

```
{moda moda_1 moda_2 modb_1}
```

Example 2

The following example overwrites the default module_uniquification_suffix of "_#" to "_uniq#" so that it is easier to identify which modules were uniquified:

uniquify_instances [get_instances u2/*-of_type design] -module_unification_suffix _uniq#

```
{moda moda_uniq1 moda_uniq2 modb_uniq1}
```

Example 3

The following example appends to the *my_design_module* collection the collection of modules returned by the uniquify operation. Note that by definition, uniquify_instances will return a collection with a different module object for each instances object because uniquified modules are only ever instantiated once.

append_to_collection *my_design_modules*
[uniquify_instances [get_instances u2/* -of_type design]]

Related Commands

create_instance
delete_instances
get_common_parent_instance
get_common_parent_instance
rename_instance
replace_instances

unmark_display_instances

Context: all contexts

Mode: setup, analysis

Usage

unmark_display_instances {***gate_id#*** | ***instance_name*** | ***pin_pathname***}... | **-All** | **-Selected** [-Display {DEBug | DESign}]

Description

Returns the specified marked instances to their original color. Color marking occurs with the mark_display_instances command.

Tip: If a gate is selected, any highlighting or marking applied to the gate is not visible until the gate is unselected.

Arguments

- ***gate_id#***

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

- ***instance_name***

  A repeatable string that specifies the name of an instance to unmark display instances.

- ***pin_pathname***

  A repeatable string that specifies the name of a pin whose gate you want to unmark display instances.

- **-All**

  A switch that specifies to unmark all the gates in the design.

- **-Select**

  A switch that specifies to unmarkll currently selected gates.

- -Display {DEBug | DESign}

  A switch and literal pair that specifies the window for which to apply the unmark_display_instances command. The following literals can be used with the -Display switch:

  DEBug
  Applies the settings to the Debug window. This is the default behavior.

  DESign
  Applies the settings to the Design window.

Examples

The following example unmarks two instances by specifying the instance name of one and the gate identification number (85) of the other:

unmark_display_instances /i$142 85

Related Commands

mark_display_instances
select_display_instances
open_visualizer

unregister_attribute

Context: unspecified, all contexts

Mode: all modes

Usage

unregister_attribute **-name** ***attribute_name*** [-object_types *type_list*] [-silent]

Description

Unregisters a user-defined attribute that was registered using the register_attribute command.

Following de-registration, the attribute is unavailable for use.

Pre-defined attributes cannot be de-registered.

If attribute_name does not exist, or if a design object type in type_list does not exist, an error message displays unless the -silent is specified.

Arguments

- **-name** ***attribute_name***

  A required switch and value pair that specifies the name of the attribute to de-register.

- -object_types *type_list*

  An optional switch and value pair that specifies the object type to which the new attribute can be assigned. The type_list string is a Tcl list that contains one or more of the following object types: module, instance, port, pin, or net. If an object type is not included in the type_list, the new attribute cannot be assigned to that object_type.

  If this argument is not specified, by default, the new attribute is deregistered for all object types.

- -silent

  An optional switch that suppresses error messages if the given attribute_name is not registered.

Example

The following example de-registers the attribute added_by for only the net object type.

```
unregister_attribute -name added_by -object_type net
```

Related Commands

get_attribute_list
get_attribute_option
get_attribute_value_list
register_attribute
report_attributes
reset_attribute_value
set_attribute_options
set_attribute_value

unregister_tcl_command

Context: unspecified, all contexts

Mode: all modes

Usage

unregister_tcl_command ***command_name***

Description

Unregisters the command called ***command_name*** that was previously registered using the register_tcl_command.

An error is generated if the command was registered before the last lock_current_registration command invocation. Commands loaded as part of a plugin are automatically locked.

Arguments

- ***command_name***

  A required string that specifies the command to unregister.

Example

The following example unregisters the commands ABC and DEF. The command ABC was registered prior to the last lock_current_registrarion command so it cannot be unregistered:

unregister_tcl_command ABC

```
//  Error: Command locked and can't be un-registered
```

unregister_tcl_command DEF

Related Commands

lock_current_registration

register_tcl_command

unselect_display_instances

Context: all contexts

Mode: setup, analysis

Usage

unselect_display_instances {{***gate_id#*** | ***instance_name*** | ***pin_pathname***} ... | **-All**}
[-Display {DEBug | DESign}]

Description

Unselects the specified objects in the Debug and Design window(s).

Arguments

- ***gate_id#***

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

- ***instance_name***

  A repeatable string that specifies the name of an instance to unselect.

- ***pin_pathname***

  A repeatable string that specifies the name of a pin associated with a gate you want to unselect.

- **-ALL**

  A switch that unselects all currently displayed gates.

- -Display {DEBug | DESign}

  A switch and literal pair that specifies the window in which this command selects objects. The following literals can be used with the -Display switch:

  DEBug
  Unselects the specified objects in the Debug window. This is the default behavior.

  DESign
  Unselects the specified objects in the Design window.

Example

The following example unselects an object identified by its instance name in the Debug window:

```
unselect_display_instances /i_top/i144
```

Related Topics

open_visualizer

select_display_instances

unsetenv

Context: unspecified, all contexts

Mode: all modes

Usage

unsetenv ***variable_name***

Description

Unsets a shell environment variable within the tool environment.

The unsetenv command unsets a shell environment variable within the current tool environment. When you only need a variable unset within the tool, this command saves you the time and trouble of exiting, issuing the UNIX unsetenv command in the shell, and re-invoking.

Note

A shell variable you unset with this command is unset only with respect to the current tool session. The command does not alter the value of the variable in the invocation shell.

To set or change the value of a variable, use the setenv command. To view the value of the variable, use the system command to pass an appropriate shell command (**echo** or **env** for example) to the operating system for execution.

Arguments

- ***variable_name***

  A required string that specifies the name of the variable.

Example

The following example unsets the SSH_AGENT_PID environment variable within the current tool environment, then confirms that it is unset:

```
system echo $SSH_AGENT_PID
66216
unsetenv  SSH_AGENT_PID
system echo $SSH_AGENT_PID
```

Related Commands

setenv

system

update_implication_detections

Context: dft -edt, patterns -scan

Mode: analysis

Usage

update_implication_detections

Description

Performs an analysis on the undetected and possibly detected faults to determine if any of those faults can be classified as detected-by-implication.

The update_implication_detections command is automatically issued after the simulate_patterns command executes a fault simulation when using external patterns or when using internal patterns in non-Atpg. The update_implication_detections command is also automatically issued as part of the create_patterns command.

By invocation default, only scan-path-associated faults for the detected-by-implication classification are analyzed. The following faults are classified as detected-by-implication when the update_implication_detections command is issued:

- A stuck-at-1 fault on the set input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the output.
- A stuck-at-1 fault on the reset input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-0 fault on the output.
- A stuck-at-0 fault on a clock input line of a transparent latch, scan latch, scan flip flop, shadow, copy, or sequential cell when the tool detects both the stuck-at-0 and stuck-at-1 faults for the associated data line.
- A stuck-at-0 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy or sequential cell when the tool detects the stuck-at-0 fault and no other ports can capture a 1.
- A stuck-at-1 fault on a data input line of a transparent latch, scan latch, scan D flip-flop, shadow, copy, or sequential cell when the tool detects the stuck-at-1 fault on the data output and no other port can capture a 0.

Related Commands

report_faults

verify_patterns

Context: patterns -scan, patterns -scan_diagnosis

Mode: analysis

Usage

verify_patterns {–**Diagnosis** |
-Create_startup_cache ***startup_cache_name*** [–Force_open] |
-Update_startup_cache ***startup_cache_name*** [–Force_open] |
-Load_startup_cache ***startup_cache_name***}

Description

Performs pattern verification for diagnosis, and creates, updates, or loads the Diagnosis Startup Cache for use with the diagnosis tool.

See "Pattern Verification and the Diagnosis Startup Cache" in the *Tessent Diagnosis User's Manual* for complete information.

Arguments

- **-Diagnosis**

  A required switch that only runs pattern verification without storing the pattern verification results into Diagnosis Startup Cache. The diagnosis tool outputs the pattern verification results to stdout and in the tool's transcript.

  If the pattern verification passes, then Tessent Diagnosis does not re-run pattern verification following the diagnose_failures command, unless there changes to the current pattern set.

- **-Create_startup_cache** ***startup_cache_name***

  A required switch and string pair that runs pattern verification and creates a new Diagnosis Startup Cache. If *startup_cache_name* already exists, then the tool produces an error.

  If the pattern verification passes, then the tool creates a new Diagnosis Startup Cache with the given filename. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not create the Diagnosis Startup Cache.

- **-Update_startup_cache** ***startup_cache_name***

  A required switch and string pair that loads an existing Diagnosis Startup Cache and compares the contents of the Diagnosis Startup Cache (netlist, patterns, and so on) to this information in the current Tessent Diagnosis session. If *startup_cache_name* does not exist, then the tool produces an error.

  If the specified Diagnosis Startup Cache does not contain the current pattern sets or does not match the current tool version, then the tool runs pattern verification.

  If the pattern verification passes, then the tool updates the specified Diagnosis Startup Cache. If the pattern verification does not pass, then the tool outputs the pattern verification results to stdout and does not update the Diagnosis Startup Cache.

- **-Load_startup_cache *startup_cache_name***

  A required switch and string pair that loads an existing Diagnosis Startup Cache. If *startup_cache_name* does not exist, then the tool produces an error and exits.

  If the specified Diagnosis Startup Cache contains the current pattern sets and matches the current tool version, then the tool loads the Diagnosis Startup Cache. Otherwise, the tool produces an error.

- -Force_open

  An optional switch used with either -Create_startup_cache or -Update_startup_cache that unlocks a locked Diagnosis Startup Cache.

Example 1

This example demonstrates creating a Diagnosis Startup Cache for future use in diagnosis with Tessent Diagnosis.

In the Tessent Diagnosis scan diagnosis point tool, the following command sequence loads the test patterns, performs the pattern verification, and creates a Diagnosis Startup Cache named *my_startup_cache*:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns –create_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessent Diagnosis, load the patterns and newly created Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns –load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
……
exit
```

Example 2

This example command sequence illustrates updating an existing Diagnosis Startup Cache (*my_startup_cache*) for future use by Tessent Diagnosis:

```
read_patterns pat1
read_patterns pat2 –append
read_patterns pat3 -append
```

```
verify_patterns –update_startup_cache ./my_startup_cache
exit
```

Subsequently, you re-invoke Tessent Diagnosis, load the patterns and updated Diagnosis Startup Cache (*my_startup_cache*), and perform diagnosis using the following command sequence:

```
read_patterns pat1
read_patterns pat2 –append
read_patterns pat3 -append
verify_patterns –load_startup_cache ./my_startup_cache
diagnose_failures log1
diagnose_failures log2
……
exit
```

Example 3

The following example provides the command sequence you would perform to use an existing Diagnosis Startup Cache with the Tessent Diagnosis Server:

1. Create the Diagnosis Startup Cache (*my_startup_cache*) with the following commands using the Tessent Diagnosis scan diagnosis point tool:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns –create_startup_cache ./my_startup_cache
exit
```

2. In the Tessent Diagnosis Server, add the patterns and load the Diagnosis Startup Cache (*my_startup_cache*) using the following command sequence:

```
add_pattern monitor1 pat1
add_pattern monitor1 pat2
add_startup_cache monitor1 ./my_startup_cache
...
```

Example 4

In this example, Tessent Diagnosis does not create a Diagnosis Startup Cache or re-run pattern verification during diagnosis:

```
read_patterns pat1
read_patterns pat2 -append
verify_patterns –diagnosis  // assume pattern verfication passed
```

```
diagnose_failures log1 // will not re-run pattern verification
```

Related Commands

diagnose_failures

read_patterns

write_atpg_setup

Context: dft -scan

Mode: analysis, insertion

Prerequisites: Scan chains must be inserted.

Usage

write_atpg_setup ***basename*** [-Replace] [-Procfile] [-NO_merge] [-NO_Edt] [-All_internal_clocks]

Description

Writes a test procedure file and the dofile file for the existing scan chains.

You can use these files to identify the scan chains when loading the scan design into Tessent FastScan. By default, the test procedure and the dofile for newly inserted scan chains are merged with those for existing scan chains.

The order of the commands in the test procedure file determine the different levels of test procedure file information produced by this command.

- If you execute write_atpg_setup without first using the read_procfile command, a test procedure file that contains default timeplates and the procedures needed for ATPG setup (for example, shift, load_unload, and test_setup) are created.
- If you run read_procfile and load a test procedure file that contains timeplate definitions, non-scan procedure definitions (capture, ram_sequential, and others), and empty or template scan procedures, and then execute the write_atpg_setup command, a more complete test procedure file is created.

Note

If the test procedure file loaded with the read_procfile command contains scan procedures (for example, shift and load_unload), they are replaced by ones generated with the write_atpg_setup command based on test logic inserted during the current session. The tool issues a W14 warning each time it replaces a procedure. However, it uses timeplates specified in the original procedure file for the newly-generated procedures. The only exceptions are for the test_setup procedure. Instead of being replaced, the events created by the write_atpg_setup command are appended to the end of the test_setup procedure.

By default, the tool automatically detects a SoC DFT flow and writes out EDT-specific commands to the ATPG setup files.

Hybrid TK/LBIST Flow Specifics

The regular scan insertion dofile includes additional commands that specify the pin names for scan enable and BIST enable signals. The tool automatically includes LogicBIST-related

commands either X-bounding or bounding of timing exceptions paths is implemented. Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***basename***

  A required string that specifies the root name for the test procedure and dofile. The files produced are *basename*.testproc and *basename*.dofile.

- -Replace

  An optional switch that replaces the contents of a file that already exists. By default, files are not replaced.

- -Procfile

  An optional switch that creates the test procedure file. By default, a test procedure file is created.

- -No_merge

  An optional switch that prevents merging the test procedure and the dofile of the newly inserted scan chains with those of the existing scan chains. By default, the test procedures and dofiles are merged.

- -NO-Edt

  An optional switch that specifies that EDT-specific commands are *not* written to the ATPG setup files. By default, the tool automatically detects a SoC DFT flow and writes out EDT-specific commands to the ATPG setup files. For more information, see the set_edt_options command.

- -All_internal_clocks

  An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Example 1

The following example writes the test procedure, the dofile, and the new netlist for the inserted scan chains to the specified filenames:

```
add_clocks 1 clk1
add_clocks 0 clk0
set_system_mode analysis
insert_test_logic

write_atpg_setup scan -replace
write_design -output_file scan.v
```

Example 2

The following example uses the 'write_atpg_setup -all_internal_clocks' switch on a design with a single internal clock (output net is pll/out1) and a single top-level clock (clk1) to specify to write out the internally defined clock as shown in the following generated test procedure file.

```
...
 procedure shift =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 1 starts at time 0
    cycle =
        force_sci ;
        measure_sco ;
        pulse clk1 ;
        pulse pll/out1 ; //internal PLL output net is pulsed during scan
chain shifting
    end;
 end;

 procedure load_unload =
    scan_group grp1 ;
    timeplate gen_tp1 ;
    // cycle 1 starts at time 0
    cycle =
        force clk1 0 ;
        force pll/out1 0 ; //internal PLL output initialized at start of
scan unload sequence
        force scan_en 1 ;
    end ;
    apply shift 4;
 end;
```

Related Commands

insert_test_logic
read_procfile
set_edt_options
write_design
write_procfile

write_cell_library

Tools Supported: Tessent Scan, Tessent FastScan, and Tessent TestKompress

Mode: all modes

Usage

write_cell_library ***new_lib_filename*** [-REPlace]

Description

Writes out a single library file of all library files loaded by the read_cell_library command, or loaded at tool invocation with the -library switch; this single library file can subsequently be loaded by the read_cell_library command or at tool invocation with the -library switch to reproduce the internal library state that would exist if the same set of original files was loaded.

Note

Library cells that were read into the tool using the read_verilog command with associated Verilog compiler directives like `celldefine and `endcelldefine are not included in the output of the write_cell_library command.

Arguments

- ***new_lib_filename***

  A required string that specifies the location of the file to which the library contents are written in cell/model format. If this file is subsequently loaded using -library new_lib_filename at invocation, the internal library state that existed when the file was written will be effectively restored.

- -REPlace

  An optional switch that specifies to replace the contents of the file specified by the new_lib_filename argument if the file already exists.

Example 1

The following example assumes the tool is invoked with the -library switch and the library files are loaded. The write_cell_library command writes the cells/models contained in those files to the new library file new_library.atpglib. If the new_library.atpglib file already exists the file is not written.

```
$ tessent -shell -library mylib.tcelllib
SETUP> write_cell_library new_library.tcelllib
```

Example 2

The following example assumes the tool is invoked without the -library switch and the library files are loaded after invocation with the read_cell_library command. The write_cell_library command writes the cells/models contained in those files to the new library file

new_library.tcelllib. If the *new_library.tcelllib* file already exists, its contents are replaced with the new library data.

```
SETUP> read_cell_library pad_lib.tcelllib
SETUP> read_cell_library standard_lib.tcelllib
SETUP> read_cell_library my_lib.tcelllib
SETUP> write_cell_library new_library.tcelllib -rep
```

Related Commands

delete_cell_library

read_cell_library

write_config_data

Context: unspecified, all contexts

Mode: all modes

Usage

write_config_data **filename**
[-wrappers wrapper_spec]
[-show_unspecified]
[-indent *integer*]
[-original_hierarchy]
[-no_banner] [-replace]

Description

Writes out the configuration data presently in memory into a file. When using the -wrappers option, the data written to the file can be limited to some specific wrappers.

Arguments

- **filename**

  A required string option that specifies the name of the file into which to write the configuration data.

- -wrappers *wrapper_spec*

  An optional option value pair used to specify the name of a configuration wrapper, a Tcl list of names of configuration wrappers, or a collection containing one or many configuration wrapper objects. Only the specified wrapper elements are written to the file.

  Just like with the report_config_datac ommand, if wrapper_spec only contains a single wrapper of type CSV, the format of the file will be changed such that a true CSV file is written out and ready to be read in to tools that support the CSV format such as Excel. This behavior is disabled if you use the -original_hierarchy option. A true CSV file has no wrapper and the line termination character is the carriage return instead of the semicolon, as is the case in a normal configuration data file.

- -show_unspecified

  An optional option that specifies that unspecified properties and singular wrappers all show up in the written file with the unspecified properties specified to their defaut values. By default, unspecified properties and singular wrappers do not appear in the output file.

- -indent *integer*

  An optional option that specifies to change the amoutn of white spaces used for indentation. By default, sub-elements are indented by two characters with respect to their parent wrapper.-original_hierarchy

- -original_hierarchy

  By default, the wrappers specified with the -wrappers options become root wrappers in the output file.You can use this option to preserve their original hierarchy and include all parent wrappers above them. When you write out a sub-wrapper without preserving the original hierarchy, you can then read it back into any valid parent wrapper using the "report_config_data -in_wrapper" command options.

- -no_banner

  A optional Boolean option that can be used to suppress the banner that is normally written at the top of the file. The banner format is the following:

  ```
  //-------------------------------------------------
  //  File created by: Tessent Shell
  //          Version: 2014.1
  //       Created on: Mon Jan 20 12:12:28 PST 2014
  //-------------------------------------------------
  ```

  Notice that the banner is also removed when the file is output in the true CSV format. See the description of the -wrappers option for more info about the CSV format.

- -replace

  An optional Boolean switch that suppresses the error that normally is issued if the filename already exists. When the -replace option is used, the existing file is silently deleted before creating the new one.

Example

The following example shows the effect of all options.

```
> #shows the configuration data currently in memory
> report_config_data

tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
  subWrapper2 {
    prop1 : v1;
  }
}
> # Standard write to a file
> write_config_data file1
> cat file1
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------


tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
```

```
  subWrapper2 {
    prop1 : v1;
  }
}
> # Limit write to one sub wrappers
> write_config_data file2 -wrappers tmp(1)/subWrapper1
> cat file2
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------


subWrapper1 {
  prop1 : v1;
}
> # Preserve the original hierarchy
> write_config_data file3 -wrappers tmp(1)/subWrapper1 -original_hierarchy
> cat file3
//-------------------------------------------------
//  File created by: Tessent Shell
//          Version: 2014.1-prerelease
//       Created on: Mon Jan 20 12:47:09 PST 2014
//-------------------------------------------------


tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
}
> #Remove the banner. Show error when file exist and -replace is not used
> catch {write_config_data file3 -no_banner}
//  Error: File file3 can not be overwritten unless you use the option -
replace.

> write_config_data file3 -no_banner -replace
> cat file3
tmp(1) {
  subWrapper1 {
    prop1 : v1;
  }
  subWrapper2 {
    prop1 : v1;
  }
}
```

Related Commands

report_config_data

report_config_data

write_core_description

Context: patterns -scan, patterns -scan_retargeting

Mode: analysis

Prerequisites: Retargetable pattern generation is enabled in patterns -scan_retargeting context (set_current_mode -type internal).

Usage

write_core_description ***file_name*** [-current_mode_only] [-replace]

Description

Writes out the core description corresponding to the current chip level.

The core description file can be generated in the following cases:

- After ATPG for the internal mode of a core, so that information can be fed forward for pattern retargeting.
- For the top level during pattern retargeting, so it can be used for reverse-mapping of chip-level failures back to the core level for diagnosis.
- For LogicBIST in the patterns -ijtag context when generating chip-level patterns.

This command works in the -scan_retargeting context because this file is required as input for failure mapping.

Hybrid TK/LBIST Flow Specifics

Writes the description of the core for the LogicBIST mode of operation. This includes description of the core pins, scan chain information, and EDT/LogicBIST hardware registers (essentially, all information to capture the core-level LogicBIST mode operation). Refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***file_name***

  A required switch that specifies the pathname of the file to which the core description is written.

- -current_mode_only

  An optional switch that specifies to only write out the current mode of the current core and the core descriptions of any of its core instances. By default, when used in the patterns -scan_retargeting context, the command writes out the entire core wrapper of the current core, including all its mode wrappers, so that the file is self-contained when later used for diagnosis failure mapping back to the core level.

- -replace

  An optional switch that specifies to overwrite the existing file file_name.

Example

The "Retargeting Example" in the *Tessent Scan and ATPG User's Manual* demonstrates the use of this command in a design context.

Related Commands

add_core_instances
delete_core_instances
read_core_descriptions
report_core_descriptions
report_core_instances

write_core_timing_constraints

Context: dft -edt, patterns -scan

Mode: analysis

Usage

write_core_timing_constraints ***filename_prefix*** [-Replace]

Description

Creates timing (SDC) files that provide the constraints related to the scan path and ATPG settings in the core design.

If launch-off capture (broadside) transition patterns are used, fast capture constraints are generated, otherwise slow capture constraints are generated.

For more information on the timing files, see "SDC Timing File Generation" in the *Tessent TestKompress User's Manual*.

Arguments

- ***filename_prefix***

  A required string that specifies the naming prefix for the timing files.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Example

The following example outputs timing files for the scan path and ATPG settings in the core design, using *atpgm* as the filename prefix:

write_core_timing_constraints atpgm

```
//  Writing atpgm_core_shift_sdc.tcl
//  Writing atpgm_slow_capture_sdc.tcl
```

Related Commands

write_edt_files

write_design

Context: all contexts

Mode: all modes

Usage 1: Creating interface modules

write_design {-output_file *filename*} [-modules *modules_list*] -interface [-verbose] [-replace]

Usage 2: Writing out created modules in rtl context

write_design {-output_file *filename*}[-created | -modules *modules_list* [-hierarchical]]
[-exclude_modules *exclude_modules*][-verbose] [-replace]

Usage 3: Writing out read-in modules in rtl context

write_design {-output_directory *directory* | -original_location [-extension_suffix *suffix*]}
[-modified | -all_read | -modules *modules_list* [-hierarchical]]
[-exclude_modules *exclude_modules*] [-verbose] [-replace]

Usage 4: Writing out read-in or created modules in no_rtl context

write_design {-output_file *filename*} [-modified | -created | -modules *modules_list*
[-hierarchical]] [-exclude_modules *exclude_modules*][-verbose] [-replace]

Usage 5: Writing out the graybox view of the current design

write_design {-output_file *filename*} [-graybox] [-verbose] [-replace]

Description

Writes the current design in Verilog netlist format to the specified file.

Use the -modules switch to write only specific modules. The default is to write the top module with all the modules in the hierarchy below except the created modules and the modules below them.

Note

The write_design command is disabled after you read a flat model using the read_flat_model command.

Usage 1: This usage case writes out the interface definition of the specified modules only. The port and parameter definitions are saved but everything else is deleted from the module; this has no impact on the module definition in memory and only affects the content of the file being written out. If you don't specify the -modules option, the interface of the current design is written out.

Verilog modules originally defined inside `celldefine and `endcelldefine directives are not included in the output netlist. For more information about using `celldefine to define cell boundaries, refer to the "Fault Locations" discussion in the *Tessent Scan and ATPG User's Manual*.

Usage 2: This usage case writes out the created modules in the rtl context. Created modules are modules created using the create_module command; these modules do not have a source file associated with them so you must specify one using the -output_file option. You can write all created modules into a single file using the -created option; this only works if all of your created modules have a common language because modules from different languages (VHDL versus Verilog) must be written into different files.

If you do not specify either -modules or -created options, the tool infers the following: "-modules [get_current_design] -hierarchical." In this case, the tool generates an error if the current design is not a created module. When you specify the -modules option, you only need to specify the created modules; otherwise, an error is generated.

When you specify the -hierarchical option, read-in modules are automatically skipped.

Usage 3: This usage case writes out read-in modules in the rtl context. Because RTL can contain many directives that affect how things are later compiled, the tool preserves the content and ordering of the RTL files when writing them out. For this reason, you can only write out read-in modules using either the -output_directory or the -original_location option. The use of the -output_file option is not allowed for read-in modules in the rtl context.

You can use the -all_read option to decompile and write all files previously read with read_verilog or read_vhdl, even if they contain no module definitions such as files containing only the definitions of System Verilog interfaces, `define statements, or files containing no modules instantiated below the current design.

You can use the -modified option to quickly write out only the modified modules, but if you also specify created modules, the tool does not write those out. However, you can write out the created modules using a separate write_design command invocation with the -output_file option specified.

If the -modules or -modified options are not specified, the tool infers the following: "-modules [get_current_design] -hierarchical." In this case, the tool generates an error if the current design is not a read-in module. When you specify the -modules option, you can only specify the read-in modules.

When you specify the -hierarchical option, the tool skips created modules.

Usage 4: This usage case writes out read-in or created modules in the no_rtl context. In the no_rtl context, the design modules are written out with all compiler directives removed. The dependencies on the file order is thus removed, allowing the concatenation of any modules into common files. You can merge both read-in and created modules into the same file.

In the no_rtl context, the recommended use model is to write out the entire design hierarchically into a single file which is the default behavior when the -modules, -created or -modified option are not specified.

Usage 5: This usage case writes out the graybox view of the current design into a single file. Only those instances and nets that have the in_graybox attribute set to true are kept in this file. All other nets and instances are removed. You can set this attribute using the set_attribute_value command but this attribute is normally populated using the analyze_graybox command.

Arguments

- -output_file *filename*

  A switch and string pair that specifies the name of the file in which to write all specified modules.

- -output_directory *directory*

  A switch and string pair that specifies the directory in which to write all specified modules. All files containing at least one of the specified modules will be written out. Sub-directories within the specified output directory will be created if the read-in files were stored in separate directories in order to avoid file name collisions.

- -original_location

  A switch that specifies to write out the specified modules in the same directory that they were read in from. All files containing at least one of the specified modules will be written out. The file name and directory location of the source file will be used except that the default or specified -extension_suffix value will be appended to the file extension.

  This option preserves the same file and directory structure as the original RTL description. The modified RTL files can be loaded into the next tool using the original -y options by simply modifying the extension search list. You must, however, have write access in the source directories. The source file can be write protected but not the source directory.

- -modules *module_list*

  An optional switch and value pair that specifies the names of the modules to write out. The *module_list* value is a single module name, a Tcl list of module names, or a collection of modules as returned by the get_modules command.

  In the rtl context when using multiple logical libraries and vhdl architectures, use the get_modules commands to select modules from a specific library or architecture. A module name in the module_list is only looked for in the default library with the default architecture.

- -hierarchical

  An optional switch that writes out all child modules below the specified modules. This switch can only be used with the –modules switch.

- -exclude_modules *exclude_modules*

  An optional switch and value pair that excludes modules from being written out. This argument is useful only when the -hierarchical option is inferred or specified. All modules below these excluded modules are also excluded if they are not specified in the -modules list or found below other non-excluded modules.

  In the rtl context, when using multiple logical libraries and vhdl architectures, use the get_modules command to select modules from specific libraries or architectures. A module name in the module_list is only looked for in the default library with the default architecture.

- -modified

  An optional switch that limits the written modules to those that were modified in insertion mode. Specifying this option is equivalent to using "-modules [get_modules -filter is_modified]."

- -all_read

  An optional switch that decompiles and writes out all files previously read with read_verilog or read_vhdl, even if they contain no modules. This switch is available only in the rtl context.

- -created

  An optional switch that specifies to limit the written modules to only those *created*. Specifying this option is equivalent to using "-modules [get_modules -filter is_created]."

- -graybox

  An optional switch and string pair that specifies that the content of the output files will not contain the complete module descriptions but only those nets and instances having the in_graybox attribute set to true. You can set this attribute using the set_attribute_value command but is normally populated using the analyze_graybox command.

  This switch is available only in the no_rtl context.

- -interface

  An optional switch that specifies to write out only the interface of the current design or the interfaces of the modules specified by the –modules option.

- -extension_suffix *suffix*

  An optional switch and string pair that specifies the suffix to use to distinguish the written-out file names from the read-in file names when the -original_location option is used. The -extension_suffix defaults to "_ts."

- -verbose

  An optional switch that specifies to print the name of each written file, one per line.

- -replace

  An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

Example 1

The following example writes out the current design and every other design module instantiated below it into a common file.

write_design -output_file design_out.v

Example 2

The following example generates an interface module description for the current design inside the file called mod.shell.

```
write_design -output_file mod.shell -interface
```

Example 3

The following example writes out all modules within the current design but stops on CoreA and CoreB. Separate commands are used to write out all modules within CoreA and CoreB. Notice how the first command does not specify the -modules option which means the current design is to be written out and the -hierarchical switch is inferred.

```
write_design -output_file myTop.v -exclude_modules {CoreA CoreB}
write_design -output_file CoreA.v -modules CoreA -hierarchical
write_design -output_file CoreB.v -modules CoreB -hierarchical
```

Related Commands

delete_design
set_current_design
read_verilog
read_vhdl

write_diagnosis

Context: patterns -scan_diagnosis

Mode: analysis

Usage

write_diagnosis [-SHORT] [-ENCoded] [-FORmat {TEXT | LAYout_marker | CSV}...] [-FILE ***name*** [-REPlace]] [-XMAP {V2LVS | CELL_HIERarchy}...]

Description

Writes diagnosis reports in text, comma separated values (CSV), and layout (physical) marker location formats.

If you specify write_diagnosis with no arguments, Tessent Diagnosis creates an ASCII text diagnosis report and names the file using the *failure_filename* from the diagnose_failures command and puts the results file where *failure_filename* is located.

Arguments

- -SHORT

  An optional switch that changes the diagnosis report from the layout-aware diagnosis report format to the traditional diagnosis report. Allows for backward compatibility with other tools for reading in the diagnosis report.

  Using this option does not change the computed diagnosis result including layout-aware diagnosis information, but rather suppresses the display of the layout-aware information.

Note

This option performs the layout-aware diagnosis, but suppresses the display of this information.

- -ENCoded

  An optional switch specifying encoding a suspect's pin pathname, net name, and cell name in the diagnosis output file.

- -FORmat {TEXT | LAYout_marker | CSV}

  An optional switch and single or multiple literals specifying the report's output format. By default, the write_diagnosis command creates an ASCII text version of the report if you specify no arguments.

Note

To produce the location tables (XMAP tables), you must select the default ASCII text report format (-format TEXT) for the write_diagnosis command.

If you use the -FORmat switch, then you must specify at least one of the following options:

- **TEXT** — Specifies ASCII text format.

- **LAYout_marker** — Specifies layout coordinate format.
- **CSV** — Specifies comma separated value (CSV) format.

You can specify any combination of these literals. For example, the following syntax creates both text (*example.diag*) and CSV (*example.csv*) reports:

write_diagnosis -format text csv -file example

- -FILE ***name***

  An optional switch and string pair specifying the filename (*name*) of the report. The tool uses the *name* value for the report filename, regardless of format, and identifies the format using the following filename suffixes:

  - *name*.csv — CSV format diagnosis report.
  - *name*.diag — ASCII text format diagnosis report.
  - *name*.lay — Layout markers.
  - *name*.enc.csv — Encoded CSV diagnosis report.
  - *name*.enc.diag — Encoded ASCII text diagnosis report.
  - *name*.enc.lay — Encoded layout markers.

- -REPlace

  An optional switch for overwriting an existing file.

- -XMAP {V2LVS | CELL_HIERarchy}

  An optional switch and single or multiple literals specifying the outputting of SPICE-mapped nets, Verilog module hierarchy, or both in the diagnosis report.

  If you use the -XMAP switch, then you must specify at least one of the following options:

  - **V2LVS** — Specifies outputting the SPICE-mapped nets.
  - **CELL_HIERarchy** — Specifies outputting Verilog module hierarchy.

  You can specify any combination of these switches. For example, the following syntax creates both the SPICE-mapped nets and the Verilog module hierarchy in the diagnosis report:

  write_diagnosis -XMAP V2LVS CELL_HIERarchy

Example 1

The following example creates a CSV and ASCII text diagnosis report with the filename *foo* in the *bar* directory, which must exist:

write_diagnosis -format text csv -file bar/foo

If you list the *bar* directory, then the following information is displayed:

```
% ls -l bar
...
```

```
-rwxrwxr-x   1 user account 1339 Aug 10 09:53 foo.csv
-rwxrwxr-x   1 user account 1768 Aug 10 09:53 foo.diag
```

Example 2

The following example creates encoded CSV, ASCII text, and layout markers with the filename *foo* in the *bar* directory, which must exist:

write_diagnosis -encoded -format text csv layout_marker -file bar/foo -replace

If you list the *bar* directory, the following information is displays:

```
% ls -l bar
...
-rwxrwxr-x   1 user account  1241 Aug 10 09:53 foo.enc.csv
-rwxrwxr-x   1 user account  1502 Aug 10 09:53 foo.enc.diag
-rwxrwxr-x   1 user account 40495 Aug 10 09:53 foo.enc.lay
```

Related Commands

report_diagnosis

write_edt_files

Context: dft -edt

Mode: analysis

Usage

write_edt_files ***filename_prefix*** [-Replace]
[-VErilog | -VHdl [-SIgnal_type STD_Logic | STD_Ulogic]]
[-SYnthesis_script {DC_Tcl | DC_Shell | RTL_Compiler}]
[-NO_Timing_constraints | -Timing_constraints]
[-Nortl_prefix | -RTl_prefix *prefix_string*]
[-REDuced_netlist]
[-Insertion Ts | Dc]
[-Single_module_per_file]
[-UNIque_bypass_lockup_clock]
[-EXClude_modules *list_of_modules*]
[-NO_SYNOpsys_pragma]
[-NO_WRITE_Design]
[-IJtag {Off | data_ports}]

Description

Creates the files that implement the EDT logic.

By default, if no other switches are specified, the following files are written:

- EDT logic RTL (**_edt.v*)
- A test procedure file (**_edt.testproc*)
- A dofile (**_edt.dofile*)
- A Design Compiler (DC) synthesis script (**_dc_script.scr*)
- A bypass dofile (**_bypass.dofile*)
- A bypass test procedure file (**_bypass.testproc*)

The following additional files are written out for the external flow:

- A core top file (**_edt_top.v*)
- A core blackbox file (**_core_blackbox.v*)

For more information, see "Creation of the EDT Logic" in the *Tessent TestKompress User's Manual*.

When the EDT logic is configured for multiple compression configurations, a separate test procedure and dofile are written for each configuration. For more information, see "Defining Dual Compression Configurations" in the *Tessent TestKompress User's Manual*.

By default, the language of the netlist read in is the format used for the EDT logic.

If you execute this command and you are using the internal IP location flow with Tessent Shell EDT insertion, the tool writes out all EDT files, inserts EDT IP into the design, and automatically changes the system mode from analysis to insertion.

If you execute this command and you are using the external IP location flow, the tool writes out all EDT files including the wrapper that instantiates the core design and EDT logic and remains in analysis mode. You can explicitly go to insertion mode, at which time you can further edit the core design and save the modifications. However, saving the design in insertion mode is only necessary if you have modified the design in insertion mode.

If you execute this command and you are using the internal IP location flow with DC-based EDT insertion, the tool writes out all EDT files and the DC insertion script and remains in analysis mode. If you change to insertion mode and modify the design, make sure the modifications are compatible with the DC synthesis script that was already generated.

Arguments

- ***filename_prefix***

  A required string that specifies a name prefix for the EDT logic files. The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the *filename_prefix*, and the files are written to that location.

- -Replace

  An optional switch that replaces the contents of each file that already exists.

- -VErilog

  An optional switch that writes the EDT logic in Verilog format. The tool does not support the generation of Verilog RTL when the netlist read in is in VHDL.

- -VHdl

  An optional switch that writes the EDT logic in VHDL 93 format.

- -SYnthesis_script DC_Tcl | DC_Shell | RTL_Compiler

  An optional switch and set of literals that specify to generate a synthesis script:

  DC_Tcl — The tool generates the DC synthesis script in TCL format.

  DC_Shell — The tool generates the DC synthesis script in DC_Shell language format.

  RTL_Compiler — The tool generates the RTL Compiler synthesis script in TCL format.

- -SIgnal_type STD_Logic | STD_Ulogic

  An optional switch and literal pair that specifies the type of signals and ports to use (std_logic or std_ulogic) when writing out the EDT logic in VHDL format.

Note

When writing out the top-level wrapper (**_edt_top.vhd*) and the black box description of the core (**_core_blackbox.vhd*), the std_logic and std_ulogic declaration of signals and ports in the original netlist are preserved.

- -REDuced_netlist **(internal flow only)**

  An optional switch that writes out the input netlist into two files as follows:

  - *<naming_prefix>_reduced_netlist.v* (or *.vhd*) — Reduced netlist containing only the part of the original netlist required for making necessary connections between pad terminals and EDT logic pins during synthesis. Use this file instead of the original input netlist with the synthesis script. Then use the output netlist from the synthesis run along with the *_rest_of_netlist.v* file to generate test patterns.
  - *<naming_prefix>_rest_of_netlist.v* (or *.vhd*) — Netlist file containing everything in the original netlist read into the tool except the modules written out in the *_reduced_netlist.v* file. Use the *_rest_of_netlist.v* file and the output netlist from the synthesis run to generate test patterns.

  Use this switch to create a reduced input netlist to save time during synthesis. For more information, see "Creation of a Reduced Netlist for Synthesis" in the *Tessent TestKompress User's Manual*.

Note

This switch can only be used with the *-Insertion Dc* command.

- -Single_module_per_file

  An optional switch that writes the EDT logic into multiple files with one Verilog module (or VHDL entity/architecture pair) in each file. Use this switch only if your synthesis flow requires the RTL files to be organized this way.

  The number of files written depends on the EDT logic setup. When the EDT logic is located outside the core, the top-level wrapper file may be split up if any pins share channel outputs. The DC synthesis script is modified to read in each of the RTL files individually.

Note

If set up to use the internal flow ("set_edt_options -location internal"), the tool ignores the -Single_module_per_file switch for the gate-level core netlist when you include the -Reduced_netlist switch.

The -Single_module_per_file switch causes the tool to use a different naming convention for the output files as described in Table 7-53.

Table 7-53. Single Module Per File Naming Convention

Old Name	New Name	Output File
<filename_prefix> *_edt_top.v* (or *.vhd*)	{<top_module_name> | <rtl_prefix>} *_edt_top.v* (or *.vhd*)	Top-level wrapper (external flow only)

Table 7-53. Single Module Per File Naming Convention

Old Name	New Name	Output File
<filename_prefix> *_edt.v* (or *.vhd*)	{<top_module_name> | <rtl_prefix>} *_edt.v* (or *.vhd*)	EDT circuitry
<filename_prefix> *_core_blackbox.v* (or *.vhd*)	<top_module_name> *_blackbox.v* (or *.vhd*)	Blackbox description of core (external flow only)

The additional *single module* files are named as follows:

- {<top_module_name> | <rtl_prefix>}*_edt_*<module_name>.*v* (or *.vhd*)
- {<top_module_name> | <rtl_prefix>}*_edt_pinshare_logic.v* (or *.vhd*) (external flow only)

The RTL prefix string is used when constructing the preceding names only if you include the -Rtl_prefix switch; otherwise it uses the top-module name. If the command's *filename_prefix* argument includes a pathname component, the tool still writes all files to that location even though the filename prefix may not appear in all filenames.

- -Nortl_prefix

  An optional switch that disables the default internal RTL naming feature. By default, the tool prepends the name of the top module of the netlist read in to the internal module/instances.

- -RTl_prefix *prefix_string*

  An optional switch and string pair that specifies a *prefix_string* to prepend to the module and instance names in the EDT logic RTL. The specified *prefix_string* must follow the normal naming rules for Verilog or VHDL identifiers.

- -Insertion Ts | Dc **(internal flow only)**

  Optional switch and literal pair that determines the method used to insert the EDT logic inside the design core. Options include:

  Ts — The tool inserts the EDT logic into the core netlist and writes out a Synopsys Design Compiler script to synthesize the EDT logic into the core netlist. This is the default behavior. For more information, see "Inserting EDT Logic During Synthesis" in the *Tessent TestKompress User's Manual*.

Note

You must enter all commands for determining configuration, test coverage, and data volume before saving out the EDT logic files. After the composite netlist is written, the tool moves to insertion mode and those commands are not available. For more information, see "ATPG Baseline Generation" in the *Tessent TestKompress User's Manual*.

Dc — Writes a Synopsys Design Compiler script to insert and synthesize the EDT logic inside the core netlist. For more information, see "Synthesizing the EDT Logic" in the *Tessent TestKompress User's Manual*.

- -NO_Timing_constraints | -Timing_constraints

  An optional switch that determines if SDC timing files are output. By default, no timing files are output. If launch-off capture (broadside) transition patterns are used, fast capture constraints are generated, otherwise slow capture constraints are generated. For more information, see "SDC Timing File Generation" in the *Tessent TestKompress User's Manual*.

- -UNIque_bypass_lockup_clock

  An optional switch that individually routes the clock pin of all scan cells involved with bypass lockup cells to the EDT logic. The clock pins created on the EDT logic boundary are named *edt_bypass_clk<n>* instead of top-level clock names.

- -EXClude_modules *list_of_modules* **(DC-based insertion only)**

  An optional switch and string pair that specifies to exclude from the complete EDT logic-inserted netlist, those modules specified by the *list_of_modules* argument; the tool excludes the modules specified by *list_of_modules* and any sub-module used by one of the specified modules, unless the sub-module is also used by another module that is to be written out. The *list_of_modules* is a list of strings that are separated by spaces.

  In the internal IP location flow with DC-based insertion, the tool writes out a Synopsys Design Compiler script to make EDT connections and to write out the modified design. Without the -EXClude_modules switch, the netlist is complete and contains a description of all the modules present in the input netlist. This switch excludes from the netlist the descriptions of the modules specified by the *list_of_modules* argument.

  During pattern generation, you need to provide the complete view of the EDT-inserted design including the module descriptions you excluded from the IP-creation output netlist with this command.

Note

If any of the modules excluded by this switch need to be modified during IP insertion, a warning message is issued but the modified modules are written out.

- -NO_SYNOpsys_pragma

  An optional switch that specifies to *not* write out pragmas when the tool writes out the RTL description of EDT logic in either Verilog or VHDL. By default, when synthesis targets Synopsys Design Compiler (-SYnthesis_script DC_Tcl | DC_Shell), a pragma is written indicating to the synthesis tool that it is a reset signal and should not be merged with other signals.

- -NO_WRITE_Design

  An optional switch that directs the tool to not write out an EDT-inserted netlist when this command is executed. In insertion mode, you can edit the netlist further and save the design.

You must save the design using the write_design command, and you must name the file using the naming convention *<filename_prefix>_edt_top_rtl.v* which is referenced by the synthesis script.

- -IJtag Off | Data_ports

  An optional switch and value pair that directs the tool to generate and save IJTAG files that describe the static configuration inputs of the TestKompress IP. These static configuration inputs set, enable, or disable certain features of the TestKompress IP: edt bypass, single chain bypass, low power and edt configuration.

  For details on how to use the IJTAG files for TestKompress ATPG, see "Setting up EDT IP for IJTAG Integration" in the *Tessent IJTAG User's Manual*.

Example 1

The following external flow example writes the files for EDT logic including bypass circuitry, and uses the filename prefix, *my_design* to name the output files:

write_edt_files my_design -verilog -replace -nortl_prefix

```
// Writing my_design_edt.v
// Writing my_design_edt_top.v
// Writing my_design_core_blackbox.v
// Writing my_design_dc_script.scr
// Writing my_design_edt.dofile
// Writing my_design_edt.testproc
// Writing my_design_bypass.dofile
// Writing my_design_bypass.testproc
```

The module and instance names in the RTL generated by the preceding example all begin with the *edt_* prefix.

Example 2

The following internal flow example shows the files written when the EDT logic is placed inside the core netlist including a reduced version of the input netlist for synthesis. The files resulting from the *-Reduced_netlist* switch are in bold.

write_edt_files my_design -verilog -replace -nortl_prefix -reduced_netlist -insertion dc

```
// Writing my_design_edt.v
// Writing my_design_dc_script.scr
// Writing my_design_reduced_netlist.v
// Writing my_design_rest_of_netlist.v
// Writing my_design_edt.dofile
// Writing my_design_edt.testproc
// Writing my_design_bypass.dofile
// Writing my_design_bypass.testproc
```

write_design -output_file my_design_new.v

Example 3

Assume the output file resulting from synthesis of *my_design_reduced_netlist.v* is named *my_design_edt_top_gate.v*. The following example shows how to invoke the tool, set the context for pattern generation, and load the Verilog files:

```
tessent -shell <other arguments>
  SETUP> set_context patterns -scan
  SETUP> read_verilog my_design_edt_top_gate.v my_design_rest_of_netlist.v
  SETUP> read_cell_library atpg.lib
  SETUP> set_current_design
```

Example 4

The following internal flow example writes out EDT logic in multiple files, one Verilog module per file. The EDT logic is placed internal to the design core, and the top-module name in the design is *core1*:

```
write_edt_files results/created -single_module_per_file -verilog
```

```
//  Writing results/core1_edt_decompressor.v
//  Writing results/core1_edt_spatial_compactor_8.v
//  Writing results/core1_edt_controlled_decoder_3_to_8.v
//  Writing results/core1_edt_compactor.v
//  Writing results/core1_edt_bypass_logic.v
//  Writing results/core1_edt.v
//  Writing results/core1_edt_mux_2_to_1.v
//  Writing results/created_dc_script.scr
//  Writing results/created_edt.dofile
//  Writing results/created_edt.testproc
//  Writing results/created_bypass.dofile
//  Writing results/created_bypass.testproc
```

```
write_design -output_file results/created_new.v
```

Example 5

The following external flow example shows how the *-Rtl_prefix* switch affects the filenames when the EDT logic is placed external to the design core, the EDT logic is written in multiple files, and the design has a top module named *mtm2*:

```
write_edt_files multiple -single_module_per_file -verilog -rtl_prefix designX_blockY
```

```
//  Writing designX_blockY_edt_decompressor.v
//  Writing designX_blockY_edt_spatial_compactor_8.v
//  Writing designX_blockY_edt_controlled_decoder_3_to_8.v
//  Writing designX_blockY_edt_compactor.v
//  Writing designX_blockY_edt.v
//  Writing designX_blockY_edt_pinshare_logic.v
//  Writing designX_blockY_edt_top.v
//  Writing mtm2_blackbox.v
//  Writing multiple_dc_script.scr
//  Writing multiple_edt.dofile
//  Writing multiple_edt.testproc
```

Example 6

The following internal flow example shows the files created when the tool inserts EDT logic into the design core with the *-Insertion Ts* switch:

write_edt_files results/created -verilog -rep -insertion ts

```
//  Writing results/created_edt.v
//  Writing results/created_dc_script.scr
//  Writing results/created_edt.dofile
//  Writing results/created_edt.testproc
//  Writing results/created_bypass.dofile
//  Writing results/created_bypass.testproc
//  ---------------------------------------------------------------------
//  Begin EDT logic insertion
//  ---------------------------------------------------------------------
//  Inserting EDT logic in the design.
//  EDT logic insertion completed.
```

write_design -output_file results/created_new.v

The output file *created_edt_top_rtl.v* contains the EDT logic inserted design without the EDT logic description.

Example 7

The following external flow example writes out timing files in addition to the default EDT files and specifies *created* for the naming prefix:

write_edt_files created -timing_constraints

```
//  Writing created_edt.v
//  Writing created_edt_top.v
//  Writing created_core_blackbox.v
//  Writing created_dc_script.scr
//  Writing created_edt.dofile
//  Writing created_edt.testproc
//  Writing created_bypass.dofile
//  Writing created_bypass.testproc
//  Writing created_edt_shift_sdc.tcl
//  Writing created_bypass_shift_sdc.tcl
//  Writing created_edt_slow_capture_sdc.tcl
```

Example 8

The following internal flow example shows the files created when the tool inserts EDT logic into the design core and the *-synthesis_script* switch is specified with the *rtl_compiler* option. The file resulting from the *-synthesis_script rtl_compiler* switch and option is in **bold**.

write_edt_files created -verilog -synthesis_script rtl_compiler -replace

```
//  Writing created_edt.v
//  Writing created_rtlc_script.scr
//  Writing created_edt.dofile
//  Writing created_edt.testproc
//  Writing created_bypass.dofile
//  Writing created_bypass.testproc
//  ---------------------------------------------------------------------
```

```
//  Begin EDT logic insertion
//  ---------------------------------------------------------------------
//  Inserting EDT logic in the design.
//  EDT logic insertion completed.
//  Note: The synthesis script assumes the modified design will be saved
//  in file 'created_edt_top_rtl.v'.
//  Update the synthesis script when saving to a different file.
```

write_design -output_file created_new.v

Example 9

In this example, assume the top level of the design has the three modules *cpu_core*, *clkblk*, and *padring* instantiated. The following commands write out the top-level module of the design but exclude the *cpu_core*, *clkblk*, and *padring* modules. After executing these commands, the output netlist contains only the top module description with the instances of these three modules:

write_edt_files created -replace
write_design -output_file created_edt_top_rtl.v -exclude_modules {cpu_core clkblk padring}

Assume the input design for this example is contained in two files: *top.v*, which contains only the top-level module, and *chip.v*, which contains the other three modules. The following example invokes the tool, sets the context for IP creation, loads the Verilog files, and runs the dofile, which is assumed to include the above write_edt_files and write_design commands:

tessent -shell
SETUP> set_context dft -edt
SETUP> read_verilog top.v chip.v
SETUP> read_cell_library atpg.lib
SETUP> set_current_design
SETUP> dofile ip_creation.do

After performing logic synthesis of the EDT logic, the output file is *created_edt_top_gate.v*. The command-line invocation for pattern generation uses the synthesized EDT logic file *created_edt_top_gate.v* as well as the file *chip.v* that contains the description of the modules excluded during IP creation.

tessent -shell
SETUP> set_context patterns -scan
SETUP> read_verilog created_edt_top_gate.v chip.v
SETUP> read_cell_library atpg.lib
SETUP> set_current_design
SETUP> dofile pattern_gen.do

Example 10

In this example, the write_edt_file command writes out RTL with synopsys pragma:

write_edt_files created -replace

In Verilog, the pragma is included as a comment near the *always* block that describes the target registers as shown here:

```
/*************************************************************************
**  Module:                        m8051_edt_decompressor
```

```
**  Tessent TestKompress version: v9.1
**  IP version:                   5
**  Date:                         Thu Aug 19 14:58:39 2010
*************************************************************************/

module m8051_edt_decompressor (edt_clock, edt_update, edt_channels_in,
        edt_scan_in);
   input edt_clock;
   input edt_update;
   input [ 1:0] edt_channels_in;
   output [15:0] edt_scan_in;

   reg    [15:0] edt_scan_in;
   reg    [11:0] lfsm_vec;
   reg    [11:0] lfsm_vec_lockup;

   // synopsys sync_set_reset edt_update
   always @(posedge edt_clock)
   begin : lfsm
      if (edt_update == 1'b1) begin
         lfsm_vec <= 12'b000000000000;
      end
      else begin
         lfsm_vec[ 0] <= lfsm_vec[ 1];
         lfsm_vec[ 1] <= lfsm_vec[ 2] ^ edt_channels_in[1];
         lfsm_vec[ 2] <= lfsm_vec[ 3];
         lfsm_vec[ 3] <= lfsm_vec[ 4] ^ edt_channels_in[1];
         lfsm_vec[ 4] <= lfsm_vec[ 5];
         lfsm_vec[ 5] <= lfsm_vec[ 6] ^ edt_channels_in[1];
         lfsm_vec[ 6] <= lfsm_vec[ 7] ^ lfsm_vec[ 3];
         lfsm_vec[ 7] <= lfsm_vec[ 8] ^ edt_channels_in[0];
         lfsm_vec[ 8] <= lfsm_vec[ 9] ^ lfsm_vec[ 4];
         lfsm_vec[ 9] <= lfsm_vec[10] ^ edt_channels_in[0];
         lfsm_vec[10] <= lfsm_vec[11] ^ lfsm_vec[ 1];
         lfsm_vec[11] <= lfsm_vec[ 0] ^ edt_channels_in[0];
      end
   end

   ....
```

In VHDL, the pragma is specified as an attribute from the Synopsys library in the declarations section of the architecture description as shown here:

```
-------------------------------------------------------------------------
--  Module:                       m8051_edt_decompressor
--  Tessent TestKompress version: v9.1
--  IP version:                   5
--  Date:                         Thu Aug 19 15:01:15 2010
-------------------------------------------------------------------------

LIBRARY ieee;
LIBRARY synopsys;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE synopsys.attributes.all;
USE work.all;
```

```
ENTITY m8051_edt_decompressor IS
 PORT (
      edt_clock       : IN    std_logic;
      edt_update      : IN    std_logic;
      edt_channels_in : IN    std_logic_vector(1 DOWNTO 0);
      edt_scan_in     : OUT   std_logic_vector(15 DOWNTO 0)
   );
END m8051_edt_decompressor;


ARCHITECTURE rtl OF m8051_edt_decompressor IS
   SIGNAL lfsm_vec          : std_logic_vector(11 DOWNTO 0);
   SIGNAL lfsm_vec_lockup   : std_logic_vector(11 DOWNTO 0);

   ATTRIBUTE sync_set_reset of edt_update : SIGNAL is "true";
BEGIN

   lfsm: PROCESS (edt_clock)
   BEGIN
      IF (edt_clock = '1') AND (edt_clock'EVENT) THEN
         IF (edt_update = '1') THEN
            lfsm_vec <= (OTHERS => '0');
         ELSE
            lfsm_vec( 0) <= lfsm_vec( 1);
            lfsm_vec( 1) <= lfsm_vec( 2) XOR edt_channels_in(1);
            lfsm_vec( 2) <= lfsm_vec( 3);
            lfsm_vec( 3) <= lfsm_vec( 4) XOR edt_channels_in(1);
            lfsm_vec( 4) <= lfsm_vec( 5);
            lfsm_vec( 5) <= lfsm_vec( 6) XOR edt_channels_in(1);
            lfsm_vec( 6) <= lfsm_vec( 7) XOR lfsm_vec( 3);
            lfsm_vec( 7) <= lfsm_vec( 8) XOR edt_channels_in(0);
            lfsm_vec( 8) <= lfsm_vec( 9) XOR lfsm_vec( 4);
            lfsm_vec( 9) <= lfsm_vec(10) XOR edt_channels_in(0);
            lfsm_vec(10) <= lfsm_vec(11) XOR lfsm_vec( 1);
            lfsm_vec(11) <= lfsm_vec( 0) XOR edt_channels_in(0);
         END IF;
      END IF;
   END PROCESS lfsm;
   ...
```

Example 11

The following example shows two commands that do not include the synopsys pragma:

write_edt_files noprag1 -synthesis_script rtl_compiler

write_edt_files noprag2 -no_synopsys_pragma

Related Commands

report_edt_configurations
report_edt_pins
write_design
set_edt_options
set_edt_pins

write_failing_paths

Context: patterns -scan_diagnosis

Mode: analysis

Usage

write_failing_paths [-DIAGnosis_***report_diagnosis_report_file_name***]
[-Pattern <***number***>]
[-ENDpoints [***Cell_Ins_Name1*** ***PO_Pin_Name1*** ...]
[-OUTput ***name*** [-REPlace]]

Description

Reports, for each diagnosed symptom from a failure log, the scan pattern number, suspect(s) and paths.

You must specify performing at-speed diagnosis using the "set_design_sources -at_speed" on command before using the write_failing_paths command. The write_failing_paths command issues a warning message if at-speed diagnosis is off. If you use the following commands:

read_patterns pat.file

set_diagnosis_options -at_speed on

write_failing_paths -diagnosis_report report.diag

Then you don't have to use the diagnose_failures command at all.

Arguments

- -DIAGnosis_***report_diagnosis_report_file_name***

  An optional switch and string pair specifying the diagnosis report filename (***diagnosis_report_file_name***) from which failure diagnosis is processed.

- -Pattern <***number***>

  An optional switch and integer pair specifying the failing pattern number for which failing paths are reported.

- -ENDpoints ***Cell_Ins_Name1 PO_Pin_Name1 ...***

  An optional switch and report failing paths that end at the given cell instance names or PO pin names. The end cell instances are the state elements where at-speed failures are captured. Default: report failing paths for all cells or POs.

- -OUTput ***name***

  An optional switch and string pair specifying the filename (*name*) where failing paths are reported. The tool will report failing paths to the screen and log file.

Example Failing Path Report

```
//  Symptom = 1
//      Scan pattern: 42
//      Suspect = 1
```

```
        PATH "path1" =
        PIN /CORE/TP004/P1001/PC00W33_0/Q;
        PIN /CORE/TM00T/AL011/U117/B1  ;
        PIN /CORE/TM00T/AL011/U117/OEQ  ;
        PIN /CORE/TM00T/AL011/U39/I1  ;
        PIN /CORE/TM00T/AL011/U39/O  ;
        PIN /CORE/TM00T/AL011/AP00A00_0/D ;
        END ;
-- skipping paths "path2-4" in this example --
        PATH "path5" =
        PIN /CORE/TP004/P1001/PC00W55_0/Q  ;
        PIN /CORE/TM00T/AL011/U116/B2  ;
        PIN /CORE/TM00T/AL011/U116/OEQ  ;
        PIN /CORE/TM00T/AL011/U54/B  ;
        PIN /CORE/TM00T/AL011/U54/OB ;
        PIN /CORE/TM00T/AL011/AP00A00_0/D  ;
        END ;
//      Suspect =  2
//         "path5"
//      Scan pattern: 121
//      Suspect =  1
//      "path1"
//      "path2"
//      Suspect =  2
//         "path5"
```

The above example shows 5 failing paths for *symptom 1*, *suspect 1* at scan pattern 42. Detailed description of the PATH definition can be found in the FastScan manual. Also shown is that for same scan pattern, 42, there is only one failing path for *symptom 1*, *suspect 2*, which is equivalent to *path5* that was defined earlier. For scan pattern 121, there are two patterns for *suspect 1 of symptom 1*, and they are path1 and path2 which are defined earlier; there is one path, path5 for *suspect 2 of symptom 1*.

Example 1

This example reports warning message because this command can only used for at-speed diagnosis.

set_diagnosis_options -at_speed off

diagnose_failures chip1.fail

write_failing_paths

Example 2

This example reports the at-speed failing paths for the diagnosis result of *chip1.fail* on the screen.

set_diagnosis_options -at_speed on

diagnose_failures chip1.fail

write_failing_paths

Example 3

In this example at-speed failing paths of scan pattern 5 for *chip1.fail* and the failing paths of scan pattern 10 for *chip2.fail* are printed on the screen.

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output chip1.diag
write_failing_paths -pattern 5
diagnose_failures chip2.fail -output chip2.diag
write_failing_paths -pattern 10
```

Example 4

The at-speed failing paths of diagnosis *chip1.diag* are printed on the screen. The *-end_cell* switch reports only failing paths ending at cell "/cpu/exu/pmi/ix926," where "/cpu/exu/pmi/ix926" is the name of the cell where at-speed failures are captured.

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output results/chip1.diag
write_failing_paths  -end_cell /cpu/exu/pmi/ix926
```

Example 5

The critical paths of diagnosis results for *chip1.fail* are written to file named *chip1.path.*

```
set_diagnosis_options -at_speed on
diagnose_failures chip1.fail -output results/chip1.diag
write_failing_paths -output chip1.path
```

Related Commands

diagnose_failures

set_design_sources

write_failures

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

Mode: analysis

Usage

write_failures ***failure_filename*** [-Replace]
[{*pin_pathname* -Stuck_at {0 | 1}}…] |
[*pin_pathname* {-SLOW | -SLOW_TO_RISE | -SLOW_TO_FALL}]
[-Pdet | -Exact]
[-Max *max_failed_scan_pat_num*] [-CYCle]
[-Faulty_chain *faulty_chain_name faulty_chain_type* {-Cell *faulty_cell_id*…}…]

Description

Injects faults and/or writes failing pattern results to a file.

The resulting file can be used to perform a diagnosis of the failures with Tessent Diagnosis or Tessent TestKompress. By default, the write_failures command performs a good simulation of the current design and test pattern source and writes the failing pattern information to the display. write_failures also does not produce the cell paths by default.

You can also use this command to inject faults into the design, simulate them, and write the resultant failures into the failure file for diagnosis. When faults are injected, a fault simulation is run with those values. The fault is injected with respect to the scan cell that is closest to the scan output.

For failure mapping, the tool creates core-level failure files by reverse mapping the top-level failures that were obtained by applying the top-level patterns generated with scan pattern retargeting. After reading a valid top-level failure file, the command extracts core-level failures for all failing cores and saves them into separate files that are named based on *failure_filename* and the naming convention as described below. The command strips the Verilog escaped characters too.

The tool names the core-level failure file for a failing core by appending its adjusted core instance name, its core module name, and its core test mode to the given failure file basename. A triple underscore ("___") is used to separate the core failure file basename from the other appended names, which are separated by double underscores ("__"). To convert the hierarchical instance name to a valid file name, the tool replaces "/" characters in the core instance name with a period character (".").

For example, the tool would save the core-level failure file for core instance core_10 of core module MyCore, tested in internal mode, to the name *./core_flogs/fail1.top___core_10__MyCore__internal*. Likewise, the tool would save the core-level failure file for the core instance A/B/core1 of core module "MyCore, tested in internal mode, to the name *./core_flogs/fail1.top___A.B.core1__MyCore__internal*.

Arguments

- ***failure_filename***

  A required string that specifies the name of a file to write the failing pattern information to.

- -Replace

  An optional switch that directs the tool to overwrite an existing *failure_file*. Use this if the specified *failire_file* already exists, and you want to update the contents.

- *pin_pathname* -Stuck_at {0 | 1}

  An optional, repeatable string, switch, and literal triplet that specifies the location and the value of a logic failure to inject into the design as follows:

  pin_pathname — A string that specifies the pin pathname of the location to inject the fault.

  -Stuck_at 0 | 1 — A switch and literal that specifies the stuck-at value for the fault. The stuck-at options include:

  0 — *stuck-at-0* fault

  1 — *stuck-at-1* fault

- *pin_pathname* {-SLOW | -SLOW_TO_RISE | -SLOW_TO_FALL}

  An optional repeatable string and switch pair that injects single transition fault to support at-speed diagnosis using the following criteria:

  pin_pathname — A string that specifies the pin pathname of the location to inject the transition fault.

  -SLOW — A literal that specifies failure at both transitions (0-> 1 and 1->0).

  -SLOW_TO_RISE — A literal that specifies failure at the 0->1 transition.

  -SLOW_TO_FALL — A literal that specifies failure at the 1->0 transition.

  This argument supports injecting a single transition fault. Additionally, when the transition fault is injected, other faults (for example, stuck-at logic faults or chain faults) cannot be injected together, specifically this argument does not support a mixture of a transition fault with other faults.

- -Pdet

  An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected for a specified fault. By default, only the binary detections are output to the file.

- -Exact

  An optional switch that directs the tool to output a failure when it simulates an X value where a binary value is expected or a binary value where an X value is expected for a specified fault. By default, only the binary detections are output to the file.

- -Max *max_failed_scan_pat_num*

  An optional switch and integer pair that specifies a maximum number of failing patterns to write for the specified faults. When the maximum number is reached, the simulation stops. By default, all failing patterns are written.

- -CYCle

  An optional switch that writes the failure file in a cycle-based format. By default, a pattern-based format is output. For more format information, see the"Input File Requirements" chapter in the *Tessent Diagnosis User's Manual*.

- -Faulty_chain *faulty_chain_name faulty_chain_type*

  An optional repeatable switch and argument pair that specifies the names and fault types of the chain faults to inject into a design and simulate. Argument options include:

  faulty_chain_name — String that specifies the name of the faulty chain.

  fault_type — literal that specifies the fault type associated with the chain. Argument options include:

  STUCK_AT_1

  STUCK_AT_0

  SLOW_TO_FALL

  SLOW_TO_RISE

  SLOW

  FAST_TO_RISE

  FAST_TO_FALL

  FAST

- -Cell ***faulty_cell_id***

  An optional switch and repeatable integer that specifies the faulty cells within the specified faulty chain.

Example 1

The following example switches to analysis system mode, loads a test pattern source, injects a stuck-at 1 failure at the i_1006/o logic in the specified a pattern-based failure file.

set_system_mode analysis
read_patterns file1
write_failures fail1 i_1006/o -stuck_at 1

```
// failing_patterns=8 simulated_patterns=36 simulation_time=0.00 sec
```

The output file looks like:

```
   4  chain1  3  L  H //uINTR/ix476
   6  /D_OUT(0)  L  H //uINTR/ix446
   7  chain1  3  L  H //uINTR/ix356
  ...
  29  /D_OUT(1)  L  H //uINTR/ix358
```

```
  29  chain1  3  L  H //uINTR/ix362
  31  /D_OUT(1)  L  H //uINTR/ix601
  31  /D_OUT(2)  L  H //uINTR/ix602
last_pattern_tested    35
```

Example 2

The following example shows a pattern-based output file written from Tessent TestKompress (4X compression).

```
0 edt_channel1 2  H L   // chain1---2---DFF_reg0
                        // chain2---2---DFF_reg1
                        // chain3---2---DFF_reg2
                        // chain4---2---DFF_reg3
1 edt_channel1 2  L H   // chain1---2---DFF_reg0
2 output[1]       H L   // PO
......
```

Example 3

The following example shows the cycle-based output file written from Tessent TestKompress (4X compression).

```
20 ch_out1 H L //pat 0 channel1 2 chain1---2---DFF_reg0
               //                 chain2---2---DFF_reg1
               //                 chain3---2---DFF_reg2
               //                 chain4---2---DFF_reg3

......
```

Example 4

The following example reverse maps a top-level failure file into core-level failure files and saves them to the *./core_flogs* directory based on the naming convention described above.

write_failures ./core_flogs/top_flog1 -rep

```
//  command: write_failures ./core_flogs/top_flog1 -rep
//  Note: Core level faillog
//    (./core_flogs/top_flog1___logic_die_BOT_inst__logic_die__internal)
//    is created for core ( logic_die_BOT_inst__logic_die__internal ).
//  Note: Extract 1 core faillog for 1 failing core from top level
      faillog ( top_flogs/top_flog1 ).
```

Related Commands

read_failures
read_patterns
report_failures
diagnose_failures

write_fault_sites

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Faults Usage:

write_fault_sites ***filename*** [-Replace] [-All | {-Path *gate_id_begin gate_id_end*}]

Bridge Faults Usage:

write_fault_sites ***filename*** [-Replace] [-All | *net_pathname_pair* | *bridge_name…* | *net_pathname…* | *instance_name…*] [-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-NOEQ]

Description

Writes path definitions or bridge entries from the internal list to a file.

The write_fault_sites command writes the specified fault sites from the internal list to a file. Depending on the active fault model, fault sites may be defined with path definitions or bridge entries. Fault sites are loaded into the internal list with the read_fault_sites command.

For more information on bridge faults and the bridge fault definition file, refer to "The Static Bridge Fault Model" in the *Tessent Scan and ATPG User's Manual*.

For more information on path delay faults and the path definition file, refer to "Creating a Path Delay Test Set" in the *Tessent Scan and ATPG User's Manual*.

By default, this command writes out the loaded fault attributes.

Path Delay Specifics

The write_fault_sites command also writes unambiguous paths that you previously added to the internal path list using the add_ambiguous_paths command. When writing derived unambiguous paths, write_fault_sites cannot list a gate if it does not have a pin pathname associated with it. For some designs, gates without associated pin pathnames could be fairly common. Therefore, refer to the written unambiguous paths for launch and capture points, but be aware the listing may not show all the gates in between.

Note

If you get ambiguous path warnings, then add_ambiguous_paths, those paths when written out lose the information about which of the ambiguous paths they were. Consequently, you cannot reload them and get back to the same set of paths.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool should write the path definition or bridge entry information.

- -Replace

  An optional switch that replaces the contents of the file, *filename*, if it already exists.

- -All

  An optional switch that displays all fault definitions for the currently loaded path definitions or bridge entries. This is the default.

- -Path *gate_id_begin gate_id_end*

  An optional switch and two-integer triplet that specifies a particular path or portion of a path whose definition you want to write. Use this argument to write_paths that were not defined in a path definition file and, therefore, were not loaded using the read_fault_sites command.

  The two integers specify two gate identification numbers that indicate the beginning and end of the path. The path begins at *gate_id_begin* and ends at *gate_id_end*.

  The value of *gate_id_begin* or *gate_id_end* is the unique gate identification number that the tool automatically assigns to every gate within the design during the model flattening process.

- *net_pathname_pair*

  A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  A repeatable string that specifies the name of a specific bridge entry.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname contained in a bridge entry.

- *instance_name*

  A repeatable string that specifies an instance pathname contained in a bridge entry.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

- -NET_pathname

  An optional switch that specifies to write net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that specifies to write pin names for bridge entries instead of net pathnames.

- -NOEQ

  An optional switch that writes only the representative fault information: net pair and fault categories.

Example 1

The following path delay example writes to the file *my_path_definitions* using the path definition file format, the pins in the specified path:

write_fault_sites /user/design/my_path_definitions -path 180 178

The following shows an example of the contents of a path definition file:

```
PATH "path0" =
    PIN /I$6/Q + ;
    PIN /I$35/B0 + ;
    PIN /I$35/C0 + ;
    PIN /I$1/I$650/IN + ;
    PIN /I$1/I$650/OUT - ;
    PIN /A_EQ_B + ;
END ;
```

Example 2

The following bridge example reports all bridge entries in the current fault site list, then writes one entry (including its attributes) to the file *my_bridge_definitions* using the bridge fault definition file format:

report_fault_sites

```
BRIDGE {
  NET1 = "/G5";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U7.BRIDGE";
}
BRIDGE {
  NET1 = "/G4";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U5.BRIDGE";
}
BRIDGE {
  NET1 = "/G11";
  NET2 = "/G10";
  FAULTS = {DS, DS, DS, DS};
  NAME = "U0.BRIDGE";
}
```

write_fault_sites /user/design/my_bridge_definitions U5.BRIDGE -name

The following shows the contents of the resultant *my_bridge_definitions* file:

```
VERSION    1.1

FAULT_TYPE="BRIDGE_STATIC_4WAY_DOM"

BRIDGE {
```

```
    NET1 = "/G4";
    NET2 = "/G10";
    FAULTS = {DS, DS, DS, DS};
    WEIGHT = 1.0000;
    DISTANCE = 4.96575000e-06 um;
    PARALLEL_run = 7.30000000e-05 um;
    X_COORDINATE = 174500;
    Y_COORDINATE = 570000;
    LAYER = "1";
    NAME = "U5.BRIDGE";
  }
```

Related Commands

add_ambiguous_paths
add_faults
delete_fault_sites
read_fault_sites
report_fault_sites

write_faults

Context: dft -edt, patterns -scan

Mode: analysis

Usage

Path Delay Usage : write_faults ***file_name*** [-Replace] [-Both | -Rise | -Fall]
[-All] [-Class *fault_class*[.*fault_subclass*]] [*object_pathname*] [-NO_Subclass]
[-VErilog | -VHdl] [-FORmat { Basic | Mtfi }]

Bridge Usage: write_faults ***file_name*** [-Replace] [-NO_Subclass] [-FORmat { Basic | Mtfi }]
{ [-ALl | *net_pathname_pair* | *bridge_name* ... | *net_pathname* ... | *instance_name* ...]
[-Class *fault_class*[.*fault_subclass*]]
[-NAME | -SINGle] [-NET_pathname | -PIN_pathname] [-NOEQ]
[-VErilog | -VHdl] [-CLOCK_Domains { ALL | *clock_pathname* ... }
[-NO_EQUivalent_clocks] [-INCLUDE_INTER_NORACE_clock_domains]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] } |
{ [-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls] }

Toggle/Iddq Usage : write_faults ***file_name*** [-Replace] [-NO_Subclass]
[-FORmat { Basic | Mtfi }]
{ [-Class *fault_class*[.*fault_subclass*]...] [-NOEQ] [-Stuck_at [01 | 0 | 1]]
[*object_pathname* ...] [-Hierarchy *integer*] [-Min_count *integer*]
[-VErilog | -VHdl] [-All | -Protected_only | -Unprotected_only] [-CEll_name]
[-CLOCK_Domains { ALL | clock_pathname ... } [-NO_EQUivalent_clocks]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] }
| { [-SCAN_Enable] [-CLOCK_Cones] [-IO] [-ASYnchronous_controls] }
| { -UNlisted }

Stuck/Transition Usage : write_faults ***file_name*** [-Replace] [-NO_Subclass]
[-FORmat { Basic | Mtfi }]
{ [-Class *fault_class*[.*fault_subclass*]...] [-NOEQ] [-Stuck_at [01 | 0 | 1]]
[pin_pathname ...] [-Hierarchy *integer*] [-Min_count *integer*]
[-VErilog | -VHdl] [-All | -Protected_only | -Unprotected_only] [-CEll_name]
[-CLOCK_Domains { ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]]
[-CAPture_procedures { ALL | *capture_procedure_name* ... }] [-DElay_data]
[-TIMING_CRITICAL] } | { [-SCAN_Enable] [-CLOCK_Cones] [-IO]
[-ASYnchronous_controls] } | { -UNlisted }

UDFM Usage : write_faults ***file_name*** [-Replace] [-NO_Subclass] [-FORmat Mtfi]
{ **-All** | -Class *fault_class*[.*fault_subclass*]... | *instance_pathname* | [-Hierarchy *integer*]
[-NOEQ] [-CLOCK_Domains { ALL | *clock_pathname* ... } [-NO_EQUivalent_clocks]
[-INCLUDE_INTER_NORACE_clock_domains]] { [-Udfm_type *name*]
[-INstance *name*] [-CEll *name*] [-MOdule *name*] } } | { -UNlisted }

Description

Writes fault information from the current fault list to a file.

For more information about the MTFI file format, refer to the *Tessent Scan and ATPG User's Manual*.

The write_faults command is similar to the report_faults command, except that the data is written into a file. You can review or modify the file and later load the information into the fault list with the read_faults command. You can use the optional arguments to narrow the focus of the report to only specific stuck-at or transition faults that occur on a specific object in a specific class. If you do not specify any of the optional arguments, write_faults writes information on all the known faults to the file.

The file contains the following columns of information for each fault:

- fault value—The fault value may be either 0 (for stuck-at-0 or slow-to-rise transition faults) or 1 (for stuck-at-1 or slow-to-fall transition faults).
- fault code—A code name that indicates the lowest-level fault class assigned to the fault.
- fault site—The pin pathname of the fault site.
- cell name (optional)—The name of the cell corresponding to the fault (included only if you used the -Cell_name argument).

You can use the -Hierarchy option to write a hierarchical summary of the selected faults. The summary identifies the number of faults in each level of hierarchy whose level does not exceed the specified level number. You can further specify the hierarchical summary by using the -Min_count option which specifies the minimum number of faults that must be in a hierarchical level before writing.

You can choose to display either collapsed or uncollapsed faults by using the set_fault_mode command.

Note that this command can write specific AU.TC and AU.PC faults when you previously issue the "report_statistics -detailed_analysis" command. If you want to report specific AU.TC faults, you must specify gate IDs instead of the full pin pathname. If you want to report specific AU.PC faults, you must specify the full pin pathname. See Example 5 for an example of writing specific AU.TC and AU.PC faults.

Arguments

- ***file_name***

  A required string that specifies the name of the file where the tool is to write the fault information. The file_name must be the first argument.

- -Replace

  An optional switch that replaces the contents of *file_name* if the file already exists.

- -Class *fault_class*[.*fault_subclass*]

  An optional switch and literal pair that specifies the fault class or fault class/sub-class to write. You should use the code when specifying a fault class or sub-class. If you do not use this switch, the default is to write all fault classes.

  Table 7-26 on page 1325 lists the fault classes. Table 7-50 on page 1921 lists the fault sub-classes.

  This command can write specific AU.TC and AU.PC faults when you previously issue the "report_statistics -detailed_analysis" command. For AU.TC faults you must specify gate IDs rather than the full pin pathname. For AU.PC faults, you must specify the full pin pathname. For more information, refer to "Example 5" on page 2086.

- -FORmat [Basic | Mtfi]

  An optional switch and literal pair that specifies the format of the fault file. By default, the command writes out in MTFI format. For more information about MTFI, refer to the "MTFI File Format" chapter in the *Tessent Scan and ATPG User's Manual*.

- -UNlisted

  Writes only the unlisted fault data. If there are multiple grayboxed instances in the design, this switch write accumulated fault numbers.

- -NO_Subclass

  Prevents the display of fault sub-classes after the fault code. The default is to display fault sub-classes when available.

- -Stuck_at 01 | 0 | 1

  An optional switch and literal pair that specifies the stuck-at or transition faults you want to write. The choices are as follows:

  01 — A literal that specifies for the tool to write both stuck-at-0 and stuck-at-1 faults for stuck-at faults; or to write both slow-to-rise and slow-to-fall faults for transition faults. This is the default.

  0 — A literal that specifies to write only the stuck-at-0 faults (slow-to-rise faults for transition faults).

  1 — A literal that specifies to write only the stuck-at-1 faults (slow-to-fall faults for transition faults).

- -All

  An optional switch that writes all faults, protected as well as unprotected, on all model, netlist primitive, and top module pins to the file. This is the default.

- *object_pathname*

  An optional repeatable string that specifies the list of pins, instances, or delay paths whose faults you want to write.

- -Hierarchy *integer*

  An optional switch and integer pair that specifies the maximum fault class hierarchy level for which you want to write a hierarchical summary of the faults.

- -Min_count *integer*

  An optional switch and integer pair that you can use with the -Hierarchy option and that specifies the minimum number of faults that must be in a hierarchical level to write a hierarchical summary of the faults. The default is 1.

- -NOEQ

  - For stuck-at, transition, toggle, iddq, and udfm faults only, an optional switch that specifies for the tool to write the fault class of equivalent faults. When you do not specify this switch, the tool writes an EQ as the fault class for any equivalent faults. This switch is meaningful only when the set_fault_mode command is set to Uncollapsed.

  - For bridge faults only, an optional switch that writes only the representative bridge fault information: the net pair and fault categories.

- -Both | -Rise | -Fall

  An optional switch that specifies which faults to write for each path already added via the Add Paths command. Use these switches only for path delay faults.

  -Both - An optional switch the specifies to write both the slow-to-rise and slow-to-fall faults. This is the default.

  -Rise - An optional switch that specifies to write only the slow-to-rise faults.

  -Fall - An optional switch that specifies to write only the slow-to-fall faults.

- -VErilog | -VHdl

  An optional switch pair that output the fault paths in either Verilog or VHDL syntax, rather than using the existing netlist independent format.

- -Protected_only

  An optional switch that writes only the protected faults on all model, netlist primitive, and top module pins to the file.

- -Unprotected_only

  An optional switch that writes only the unprotected faults on all model, netlist primitive, and top module pins to the file.

- -CEll_name

  An optional switch that specifies to list, for each reported fault, the corresponding circuit element (ATPG library cell, Verilog primitive, primary input, or primary output) as summarized in Table 7-54. Cell names displayed for equivalent faults are based on the cells

associated with the equivalent faults, not the cells associated with the representative faults. This switch is valid for the stuck-at, transition, toggle and IDDQ fault models only.

Table 7-54. Name Conventions Used by write_faults -Cell_name

Corresponding Circuit Element	Listed Cell Name
ATPG library cell	Name of the ATPG library cell
Supported Verilog primitive (for a list, see Table 4-1 in the *Tessent Cell Library Manual*)	Name of the Verilog primitive
Primary input or output	“primary_input” or “primary_output”

- *net_pathname_pair*

  A string pair that specifies the net or pin pathname to bridge fault site. The first string specifies NET1, and the second string specifies NET2.

- *bridge_name*

  A repeatable string that specifies the name of a specific bridge entry.

- *net_pathname*

  A repeatable string that specifies a single net or pin pathname contained in a bridge entry.

- *instance_name*

  A repeatable string that specifies an instance pathname contained in a bridge entry.

- -NAME

  An optional switch that specifies all entered strings are bridge names.

- -SINGle

  An optional switch that specifies all the entered strings are either a single net pathname or instance pathname. The tool searches for a net first. If a corresponding net cannot be found, the tool searches for a corresponding instance.

- -NET_pathname

  An optional switch that reports net pathnames for bridge entries. This is the default.

- -PIN_pathname

  An optional switch that reports pin pathnames for bridge entries instead of net pathnames.

- -CLOCK_Domains {ALL | *clock_pathname*…}

  An optional switch and literal or repeatable string pair that specifies a list of clocks and directs the tool to write only faults that are potentially detectable using any of the specified clocks (or equivalent clocks). For a transition fault to be potentially detectable, it must be detectable when the same clock (one of the specified clocks or an equivalent) is used for launch and capture. Any other type of fault must be detectable when one of the specified clocks or equivalents is used as the capture clock. The argument choices are as follows:

ALL — A literal that specifies all the clocks in the design.

clock_pathname — A repeatable string that specifies a particular clock.

Be aware that a fault written using this switch might also be detectable by an unspecified clock.

- -NO_EQUivalent_clocks

  An optional switch that prevents the -Clock_domains switch from writing faults in equivalent clock domains.

- -INCLUDE_INTER_NORACE_clock_domains

  An optional switch used with -Clock_domains to instruct the tool to take user-defined non-race clocks into account when writing faults in the clock domain. You can use this switch in non-setup mode and when the clock restriction is set to be domain_clock.

- -CAPture_procedures {ALL | *capture_procedure_name*…}

  An optional switch and literal or repeatable string pair that specifies a list of enabled named capture procedures and directs the tool to write_faults that are potentially detectable by any of the specified procedures. The argument choices are as follows:

  ALL — A literal that specifies all enabled named capture procedures.

  capture_procedure_name — A repeatable string that specifies a particular enabled named capture procedure.

  See the set_capture_procedures command description for information about enabling or disabling named capture procedures.

- -DElay_data

  An optional switch that saves the delay faults including the slacks of the faulty site.

- -TIMING_CRITICAL

  An optional switch that saves the timing critical faults identified by the tool. Before issuing the write_faults command with this switch, you must issue the set_atpg_timing command with the -TIMING_CRITICAL switch and *timing_margin%* value.

Note

In order to use the -DElay_data or -TIMING_CRITICAL switches, you must have previously loaded an SDF file containing timing information. See "Using Timing-Aware ATPG" in the *Tessent Scan and ATPG User's Manual* for complete information.

- -SCAN_Enable

  An optional switch that specifies to only write faults that fan out to the select lines of multiplexers in the scan path. For this switch, a multiplexer is either a MUX simulation primitive or a nonprimitive type multiplexer composed of AND and OR gates. Basically, this switch writes all faults that are in the fanin cone of local scan enable signals and are dominated by them.

- -CLOCK_Cones

  An optional switch that specifies to only write faults that fan out to clock ports, Set/Reset ports or Read/Write control ports of memory elements, including RAM and ROM. Notice that this applies to any memory elements, not just scan cells.

- -IO

  An optional switch that specifies to write faults that are:

  - Only controlled by PIs — To be acted on by the -IO switch, a PI either must not be defined as a clock or, if defined as clock (or read/write control), must be constrained off during capture.
  - Only observed by POs

- -ASYnchronous_controls

  An optional switch that specifies to only write faults that fan out to Set/Reset ports of state elements and RAMs. This switch applies to a subset of the faults the -CLOCK_Cones switch writes.

- -Udfm_type ***name***

  An optional switch and string that specifies a name of a UDFM fault type. Use this option to to write the faults associated with a specified UDFM fault type to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -CEll ***name***

  An optional switch and string pair that specifies the name of a library cell. Use this option to write all faults associated with a specified library cell to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -MOdule ***name***

  An optional switch and string pair that specifies the name of a module. Use this option to write all faults associated with a specified module to the file. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

- -INSTance ***name***

  An optional switch and string pair that specifies the name of an instance. Use this option to display all faults associated with a specified instance. The name string may include any number of asterisk (*) and/or question mark (?) wildcard characters.

Example 1

The following example performs an ATPG run, then writes all the untestable faults to a file for review:

```
set_fault_type stuck
set_system_mode analysis
create_patterns
write_faults faultlist -class ut -format basic
```

The content of the *faultlist* file will look similar to the following:

```
0     UU    /ip_scan_clk
1     UU    /ip_scan_clk
0     UU    /ipt_scan_tf_en
1     UU    /ipt_scan_tf_en
0     UU    /ff0/qb
1     UU    /ff0/qb
...
```

Example 2

The following example rewrites all the untestable faults to the file, but includes the name of the cell associated with each fault in the written information:

write_faults faultlist -class ut -cell_name -format basic -replace

The file now contains an additional column with the cell information:

```
0     UU    /ip_scan_clk  primary_input
1     UU    /ip_scan_clk  primary_input
0     UU    /ipt_scan_tf_en  primary_input
1     UU    /ipt_scan_tf_en  primary_input
0     UU    /ff0/qb  dsffd
1     UU    /ff0/qb  dsffd
```

Example 3

The following command writes out a fault file in MTFI format that contains all faults:

write_faults fault_file1 -format mtfi

This is an example of a small MTFI file that could result from the previous command:

```
FaultInformation {
   version : 1;
   FaultType Stuck  {
      FaultList {
         Format : Identifier, Class, Location;
         Instance ("") {
            1, UC, "in1";
            0, DS, "in1";
            0, DS, "i1/IN0";
            1, DS, "i1/OUT";
            1, DS, "i1/IN0";
            1, AU, "i1/IN1";
            0, UC, "i5/Z";
            1, DS, "out";
            1, DI , "i5/Z";
         }
      }
   }
}
```

Example 4

The following example displays the fault data for *ds* class faults associated with library cells that start with *mux* or *an* to the *my_fault_list* file:

write_faults my_fault_list -class ds -cell mux* -cell an*

```
// Command output

FaultInformation {
   version : 1;
   FaultType (UDFM) {
      FaultList {
         Format : Identifier, Class;
         UdfmType ("intra-cell-bridges_1") {
            Cell ("MUX2") {
               Instance ("/top/i1/i4") {
                  "F2", DS;
               }
            }
            Cell ("AND3") {
               Instance ("/top/i1/i456") {
                  "F2", DS;
                  "F4", DS;
                  "F5", DS;
                  "F7", DS;
               }
            }
         }
      }
   }
}
```

Example 5

The following example shows how to write specific AU.TC and AU.PC faults:

report_statistics -detailed_analysis ;# you must first run this command

write_faults fltlist_au.tc -class au.tc 1195 ;# you must supply a gate ID rather than pin pathname

write_faults fltlist_au.pc -class au.pc mode[3] ;# you must supply a pin pathname

Related Commands

add_faults
delete_faults
read_faults
set_checkpointing_options
report_faults
set_fault_mode
set_fault_protection
set_fault_sampling
set_fault_type

write_flat_model

Context: patterns -scan, patterns -scan_diagnosis

Mode: setup, analysis

Usage

write_flat_model ***filename***
[-Password *password*] [-Command_file *command_file*]
[-Replace] [-All | -Flat]

Description

Saves the flattened circuit model, the scan trace, and all DRC-related information to a specific binary file.

Note

After saving the flattened model, you cannot add primary inputs to the model. This includes using the "add_scan_chains -internal" command.

This command enables you to quickly restore a new session to the state of a previous session based on saved check point data. This command lets you enter into analysis mode without reading the netlist, flattening, and performing DRC.

The write_flat_model command does not save pattern or fault list information in the flattened model file. However, you can use the set_checkpointing_options command to save the pattern set and the fault list periodically.

Note

If you are preparing a flat model for later use in Tessent Diagnosis, save the flat model immediately after performing ATPG (issuing the create_patterns command) and saving final patterns. For more information, refer to "Preparing the Design Netlist" in the *Tessent Diagnosis User's Manual*.

When reading in a previously-stored flattened model, the tool defaults to the same system mode as when the model was saved. It is also possible to save only the flattened model (no scan chain or DRC data) to be restored into setup mode from any system mode.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the flattened circuit model.

- -Password *password*

  An optional switch and replaceable string pair that directs the tool to embed the specified *password* in the saved file. The *password* is case sensitive and there are no restrictions on its content. Before reading in a flattened model that was saved using the -Password switch, the

tool will require you to enter the password that was provided with the switch when the model was saved.

For batch runs, you can set the MGC_DFT_FLAT_MODEL_PASSWORD environment variable to the specified password; the tool will then obtain the password from the environment variable instead of querying you for it. If the password setting of this variable is incorrect, however, the tool will revert to querying you for it.

Caution

Be sure to write down the password and store this copy in a readily accessible, secure location. This will ensure that a forgotten password is only a minor inconvenience.

- -Command_file *command_file*

  An optional switch and string pair that specifies the pathname of an ASCII file you previously created that lists commands you want the tool to permanently disable for this flattened model. You can create this file with any ASCII text editor.

  Choose one of the two keywords, **disable** or **enable**, to place by itself on the first line of the command file. On subsequent lines, list the commands to which the keyword applies, one command or command group per line. The keywords have the following meaning:

  - **disable**—Disables the commands listed after the keyword. If the design is one you read into the tool as a flattened model, any commands already disabled as a result of the previous flattening will remain disabled; omitting them from the current “disable” list will not re-enable them. All other commands the tool normally supports will be available.
  - **enable**—Disables all normally supported commands *except* those listed after the keyword. If the design is one you read into the tool as a flattened model, commands already disabled as a result of the previous flattening will remain disabled; listing them after the “enable” keyword will not re-enable them.

Note

The dofile, exit, and help commands are immune to the command file and are always available.

You specify a command group by listing just the first word or words that all commands in the group have in common. For example, you could list “set” on one line in order to specify the group of all commands that start with the word, “set.”

- -Replace

  An optional argument that lets you overwrite an existing circuit model file.

- -Flat

  An optional argument that specifies only the circuit model information will be saved, excluding all DRC related information that exists when the command is issued. This is equivalent to saving the model in Setup mode.

- -All

  An optional argument that specifies for all circuit model information, including all DRC-related information, to be saved when the command is issued. This is the default.

Example 1

The following example creates a flattened model and saves it.

create_flat_model
write_flat_model file1

Example 2

The following example uses an external command file to ensure the saved flattened model only works with the following commands:

dofile
exit
help
any command that starts with “set”
any command that starts with “write primary”

create_flat_model
write_flat_model file1 -command_file commands_for_file1
exit

```
shell> cat commands_for_file1
enable
set
write primary
```

Related Commands

diagnose_failures
create_flat_model
read_flat_model
set_checkpointing_options
set_file_compression
write_faults

write_icl

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

Mode: setup, analysis

Usage

write_icl {**-output_file *filename***} [-modules *modules_list* [-hierarchical]] [-exclude_modules *exclude_modules*] [-created] [-modified] [-replace]

Description

Writes out ICL modules created and/or read in with the read_icl command to the specified file.

This command writes out read-in and/or created ICL modules. You can merge both read-in and created ICL modules into the same file. The recommended use model is to write out the ICL hierarchically into a single file which is the default behavior when neither the -modules or -created option are specified.

Use the -modules switch to write only specific ICL modules. The default is to write the top module with all of the modules in the hierarchy below including the read-in modules and the modules below them. Use the -created option if you only want to write out the ICL modules created by ICL extraction and omit those that were read-in

Arguments

- **-output_file *filenames***

  A required switch and string pair that specifies the name of the file to which to write all specified modules.

- -modules *module_list*

  An optional switch and value pair that specifies the names of the modules to write out. The *module_list* value is a single ICL module name, a Tcl list of ICL module names, or a collection of ICL modules as returned by the get_icl_modules command.

- -hierarchical

  An optional switch that specifies to write out all child modules below the specified ICL modules. This switch can only be used with the –modules switch.

- -exclude_modules *exclude_modules*

  An optional switch and value pair that excludes ICL modules from being written out. This argument is only useful when the -hierarchical option is inferred or specified. All ICL modules below these excluded ICL modules are also excluded if they are not specified in the -modules list or found below other non-excluded ICL modules.

- -created

  An optional switch that specifies to limit the written ICL modules to only those *created*. Specifying this option is equivalent to using the following:
  “-modules [get_icl_modules -filter is_created].”

- -modified

  An optional switch that verifies whether a module has the is_modified attribute set to true; if it does, the tool writes out the module.

- -replace

  An optional switch that specifies to overwrite an existing file. When you use this switch, the error message that is normally generated if the file being written out already exists is suppressed.

Example

The following example writes out the current design and every other ICL module instantiated below it into a common file.

```
write_icl -output_file icl_out.icl
```

Related Commands

get_icl_modules
read_icl
set_current_design

write_loops

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

Mode: analysis

Usage

write_loops ***filename*** [-Replace]

Description

Writes a list of all the current loops to a file.

The write_loops command writes all feedback loops in the circuit to a file. For each loop, the file contents show whether the loop was broken by duplication. The file shows loops unbroken by duplication as being broken by a constant value, which means the loop is either a coupling loop or has a single, multiple-fanout gate. The report also includes the pin pathname and gate type of each gate in each loop.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the loops.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Example

The following example writes a list of all the loops in the circuit to a file:

```
set_system_mode analysis
write_loops loop.info -replace
```

Related Commands

report_loops

write_modelfile

Context: dft -edt, patterns -scan

Mode: analysis

Usage

write_modelfile ***filename RAM/ROM_instance_name*** [-Replace]

Description

Writes all internal states for a RAM or ROM gate into the file that you specify.

The write_modelfile command writes, in the Mentor Graphics modelfile format, all the internal states for a RAM or ROM gate into a file.

The RAM and ROM internal states are identical to the initial values. If no initial values exist, the command reports an error condition.

Arguments

- ***filename***

  A required string that specifies the name of the file to which you want to write the internal states for the RAM or ROM gate. The information is written in Mentor Graphics modelfile format.

- ***RAM/ROM_instance_name***

  A required string that specifies the instance name of the RAM or ROM gate whose internal states you want to write. The command reports an error condition if the instance contains multiple RAM/ROM gates.

- -Replace

  An optional switch that replaces the contents of the file if the *filename* already exists.

Example

The following example writes all the internal states of a RAM gate into a file for review:

```
add_write_controls 0 w1
set_system_mode analysis
create_patterns
write_modelfile model.ram /p1.ram
```

Related Commands

read_modelfile

set_ram_initialization(FS)

write_patterns

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

Mode: analysis

Usage

For patterns -scan and EDT Off

write_patterns ***pattern_filename*** [-pattern_sets [chain] [scan]]
[*format_switch*] [-Replace]
[-PROcfile *procedure_filename*] [-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair*...}] [-PARALlel | -Serial] [-NOInitialization]
[-BEgin {*pattern_number* | *pattern_name*}] [-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*] [-CEll_placement {Bottom | Top | None}] [-ENVironment]
[-TEST_Setup { ONE_Per_file | OFf | ONLy | ONE }] [-TEST_End { ONE | ONLy | OFf}]
[-EXternal] [-SAmple [*integer*]] [-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-NAme_match | -POSition_match] [-NOPadding | -PAD0 | -PAD1] [-Noz]
[-MAXloads *load_number*] [-PATtern_size *integer*] [-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*] [-VERBose]

For patterns -scan and EDT On

write_patterns ***pattern_filename*** [-pattern_sets [chain] [scan]]
[*format_switch*] [-Replace]
[-PROcfile *procedure_filename*] [-PARAMeter_file *parameter_filename*]
[-PARAMETER_List {*keyword_value_pair*...}] [-PARALlel | -Serial] [-NOInitialization]
[-BEgin {*pattern_number* | *pattern_name*}] [-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*] [-CEll_placement {Bottom | Top | None}] [-ENVironment]
[-TEST_Setup { ONE_Per_file | OFf | ONLy | ONE }] [-TEST_End { ONE | ONLy | OFf}]
[-EXternal] [-SAmple [*integer*]] [-IDDQ_file]
[-MODE_Internal | -MODE_External] [-PAD0 | -PAD1] [-Noz]
[-MAXloads *load_number*] [-PATtern_size *integer*] [-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*]
[{-EDT_Bypass | -EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External]

For patterns -ijtag

write_patterns **pattern_filename** [-pattern_sets *pattern_set_list*]
[format_switch] [**-Replace**] [-PROcfile *procedure_filename*]
[-PARAMETER_file ***parameter_filename***] [-ENVironment]
[-PARAMeter_List {*keyword_value_pair*...}]
[-TEST_Setup { ONE_Per_file | OFf | ONLy | ONE}] [-TEST_End { ONE | ONLy | OFf }]

For patterns -scan_retargeting

write_patterns ***pattern_filename*** [-pattern_sets [chain] [scan]]
[*format_switch*] [-Replace]
[-PROcfile *procedure_filename*] [-PARAMeter_file *parameter_filename*]

[-PARAMETER_List{*keyword_value_pair*...}] [-PARALlel | -Serial] [-NOInitialization]
[-BEgin {*pattern_number* | *pattern_name*}] [-END {*pattern_number* | *pattern_name*}]
[-TAg *tag_name*] [-CEll_placement {Bottom | Top | None}] [-ENVironment]
[-TEST_Setup { ONE_Per_file | OFf | ONLy | ONE }] [-TEST_End { ONE | ONLy | OFf}]
[-EXternal] [-IDDQ_file]
[-MODE_Internal | -MODE_External]
[-NAme_match | -POSition_match] [-NOPadding | -PAD0 | -PAD1] [-Noz]
[-MAXloads *load_number*] [-PATtern_size *integer*] [-MEMory_size *size_in_KB*]
[-SCAN_Memory_size *size_in_KB*] [-VERBose]

For Writing Out Hybrid TK/LBIST Flow Parallel Patterns

write_patterns ***filename*** -verilog -parallel -mode_internal [-replace]

For Writing Out Hybrid TK/LBIST Flow PatternDB Files

write_patterns ***filename*** -patdb [-replace]

Description

Saves the current test pattern set to a file in a specified format.

By default, the module name created for Verilog test benches consist of the specified design name followed by an underscore, followed by the name of the specified test procedure file followed by _ctl. All periods are converted to underscores. For example:

design_name_test_proc_file_name_ctrl

Note that you can type Ctrl-C on the command line to cancel the write_patterns operation without creating any files.

Note

The tool issues an AG11 DRC error if you attempt to write patterns in a parallel format other than Verilog or STIL using a multiple cycle shift procedure. Any serial format is fine.

Hybrid TK/LBIST Flow Specifics

Writes out parallel patterns and PatternDB files—refer to the *Hybrid TK/LBIST Flow User's Manual* for complete information.

Arguments

- ***pattern_filename***

  A required string that specifies a name for the saved test pattern file. If the filename ends with ".gz," the file is automatically compressed with the GNU gzip utility. If the filename ends with ".Z," the file is automatically compressed with the UNIX compress utility. For more information, see the set_file_compression command.

- *format_switch*

  An optional switch that specifies a format for the saved test patterns. You should always save simulation patterns in the same format as the design netlist.

ASIC vendor test data formats usually support only a single timing file. You can specify the test timing that each ASIC vendor requires by using different timing definition files for each format.

Any time you save the pattern set in an ASIC vendor data format, you should also save the patterns in ASCII, binary, STIL or WGL format as backup. This is in case you want to read in the patterns from an external file using the read_patterns command. Only ASCII, binary, STIL or WGL formatted test patterns can be read by the ATPG tools.

Options include:

-Ascii — A switch that writes test patterns in the standard ASCII format. The ASCII format includes the statistics report, environment settings, scan structure definition, scan chain functional test, scan test patterns, and the scan cell information. This is the default.

Excess load and unload values are padded with Xs. You can override this behavior with the -Nopadding switch.

If you use the -External or the -Begin and -End switches, test coverage and fault information is omitted from the ASCII pattern set.

-BInary — A switch that writes the test patterns in binary format. This option is not supported in the patterns -scan_retargeting context.

-CTl2005 — A switch that writes the patterns in CTL format. The STIL language is used, but in a more structured manner, and with the addition of CTL information. CTL is an extension to the STIL language. You can customize the output format when you utilize a parameter file. For more information, refer to the description of the -Parameter switch later in this section and to "Parameter File Format and Keywords" on page 2679.

-PDl — A switch that writes the pattern back out in PDL format. This format is supported only for patterns -ijtag context. The original PDL is converted to iRead and iWrite commands on ScanRegisters and Primary input/output pins. The scan operations needed to perform the ICL network reconfiguration are also explicitly described with iWrite and iApply commands. All PDL events associated with each pattern set are grouped into individual iProc wrappers and each iProc is named with the name of the corresponding pattern set.

-STIL2005 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0 and 1450.1. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to "Parameter File Format and Keywords" on page 2679. Note that minimum typing for this switch is -STIL2.

-STil1999 — A switch that writes the patterns in STIL format that conforms to IEEE Std. 1450.0. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to "Parameter File Format and Keywords" on page 2679.

-Verilog — A switch that writes the patterns in Verilog format. The Verilog format contains pattern information and timing information from the timing file as a

sequence of events. Saving patterns in Verilog format also causes the tool to write out a configuration (.cfg) file along with the testbench and .vec files. For example:

```
Patterns.v
Patterns.v.0.vec
Patterns.v.1.vec
Patterns.v.cfg
Patterns.v.po.name
Patterns.v.chain.name
```

The configuration file contain the following lines:

```
Patterns.v.0.vec  740
Patterns.v.1.vec  820
```

(The number after the filename is the number of data lines to read from the .vec file.)

Even though the testbench is created with each write_patterns command, you only have to compile the design and testbench .v files once. You can use additional patterns just by copying the .vec and .cfg file to the location of the compiled simulation files.

All timeplates that are currently loaded into the tool are also saved in the Verilog testbench, not just the timeplates used by the current set of patterns. If a new procedure file with different timeplates or if the netlist information is modified in any way, you have to compile the new testbench before simulating.

The default Verilog testbench supports three Verilog plusargs: STARTPAT, ENDPAT, and CHAINTEST. For more information, refer to "The Verilog Test Bench" in the *Tessent Scan and ATPG User's Manual*.

You can use the Verilog patterns to interface to a Verilog simulator such as ModelSim. You can customize the output format with a parameter file. For more information, see the description of the -Parameter switch later in this section and to "Parameter File Format and Keywords" on page 2679.

-Wgl — A switch that writes the patterns in the WGL format. The WGL format contains the waveform pattern information and any timing information from the timing file. You can use the WGL format patterns to generate test patterns in a variety of tester and simulator formats.

-Fjtdl — A switch that writes the test patterns in the Fujitsu ASIC format: FTDL-E.

-MItdl — A switch that writes the test patterns in the Mitsubishi ASCI format: MITDL.

For MITDL format there is restriction that multiple pulse timing must be a cyclical repetition of the first pulse. Consequently, multi-pulse and double-pulse timing in the procedure file only works in the MITDL output without an error if the timing fits the restrictions of the MITDL syntax. See "Timeplate Definition" in the *Tessent Shell User's Manual*.

-Mode Lsi — A switch that changes the functionality of the Verilog and WGL output formats so that the saved pattern files meet LSI Logic requirements. This switch is valid only with the -Verilog and -Wgl switches.

-TItdl — A switch that writes the test patterns in the Texas Instruments ASIC format: TDL91. You can customize the output format with the parameter file. For more information on the format of the parameter file, refer to "Parameter File Format and Keywords" on page 2679.

Note

If the design uses a single timeplate the TDL version 4.3 format is used. This removes the TIMING, END_TIMING, and SET_TIMING statements from the test pattern file. Also, the VERSION statement changes from 6.0 to 4.3. However, if the design uses multiple timeplates, then the TDL version output is 6.0.

-TSTl2 — A switch that writes the patterns in Toshiba ASIC format: TSTL2.

-SVf — A switch that writes out the IJTAG patterns in Serial Vector Format. This switch is supported only for "patterns -ijtag" context and can only be used to write IJTAG pattern sets to files.

This switch can be used only if the TAP interface is the only scan interface described in the top ICL module. An error message is issued if there is no TAP interface or there are other scan interfaces besides the TAP interface. The TCK ratio of the pattern set(s) must be 1 (the TCK ratio is set by means of the -tck_ratio switch of the open_pattern_set command).

The –test_setup and –test_end switches of the write_patterns command are taken into account. If requested, the test setup procedure and the test end procedure are also converted to SVF and placed at the beginning/at the end of the pattern file.

- -Replace

  An optional switch that replaces the contents of an existing *pattern_filename*.

- -PROcfile *procedure_filename*

  An optional switch and string that specifies the name of the test procedure file to retrieve non-scan event timing information from. This option saves test patterns using the enhanced pattern output that includes timing information from the test procedure file.

  You can specify the procedure filename with this argument or specify it before using this command by issuing the read_procfile command. You cannot use this argument with the -Ascii or -Binary formats.

  For more details about the test procedure file, including which output format switches are valid in conjunction with the -Procfile switch, refer to "Test Procedure File" in the *Tessent Shell User's Manual*.

- -pattern_sets [chain] [scan] **(patterns -scan only)**

  An optional switch and keyword pair that specifies which pattern type (scan or chain) to write from the current session. By default, the tool writes both scan and chain patterns. Note that when you specify both chain and scan patterns, the tool writes chain patterns first regardless of the order of specification.

- -pattern_sets *pattern_set_list* **(patterns -ijtag only)**

  An optional switch and string that specifies a Tcl list of pattern set names that were created using the open_pattern_set command. The tool writes patterns in the order specified in the pattern_set_list. When you don't use the -pattern_sets switch, the command writes all pattern sets in the order specified by the open_pattern_set command.

- -PARAMeter_file *parameter_filename*

  An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain pattern output formats. For more information about the format of the parameter file, refer to "Parameter File Format and Keywords" on page 2679.

- -PARAMETER_List {*keyword_value_pair*...}

  An optional switch and keyword/value pair that specifies one or more parameter file keywords and their values for use in certain pattern output formats. The keywords and values are delimited by spaces, and you must enclose the list in curly braces { }. For more information about parameter file keywords, refer to "Parameter File Format and Keywords" on page 2679.

- -PARALlel

  An optional switch that saves all scan cells in parallel. This is the default.

  In designs with scan cells, only scan pins are active during the scan shift cycles. If the tool tries to represent the state of each pin during each shift cycle, it may produce very large pattern files. Simulating the shift operations of these patterns might require a considerable amount of time if you use a different simulator. You can avoid these problems by saving all scan cells in parallel. The -Parallel switch is compatible with all formats.

Note

When EDT is On and you save parallel Verilog patterns, the data is saved with respect to the internal scan chains. These patterns are for simulation and cannot be saved as serial. But when you write patterns in parallel tester formats (for example, WGL), the tool saves the data with respect to the external scan channels. These patterns are meant to be saved in serial and cannot be used for parallel simulation.

- -Serial

  An optional switch that saves all scan cells in series. You can only use this switch with the -STil1999, -STIL2005, -TItdl (enhanced pattern output only), -Wgl, and -Verilog format type switches.

Note

You cannot use the -Serial switch and the -Scan_memory_size switch in the same command.

When you use this switch together with the -Verilog switch and the -Parameter switch in the same command, the tool checks the specified parameter file for the SIM_SHIFT_DEBUG keyword. If this keyword exists and is set to 1, every scan chain in the Verilog patterns

contain a parallel observe of the entire scan chain for *every shift*, while serially shifting the test patterns. If this keyword is set to 2 or more, the parallel observe occurs for that many shifts, not all. This format is particularly useful for debugging scan chain problems because it allows you to locate the exact scan element causing a problem in a scan chain.

Tip: Use SIM_SHIFT_DEBUG serial Verilog patterns only for debugging scan chain problems. They typically take more time to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be large). Also, because they simulate serially, the patterns take more time to simulate. Consider using the "-pattern_sets chain," -Begin, and -End switches to limit the size of the pattern file.

For more information about the parameter file and the SIM_SHIFT_DEBUG keyword, refer to"Parameter File Format and Keywords" on page 2679.

- -NOInitialization

  A switch that writes patterns without creating the initialization cycle in the pattern file. The -Noinitialization switch is valid with the following output formats:

  - STil
  - Verilog
  - WG1 (Ascii)

  The -Noinitialization switch is valid with all enhanced pattern output formats. For more information on the enhanced pattern output, see "Test Procedure File" in the *Tessent Shell User's Manual*.

Note

If DRC was run with initialization enabled (set_drc_handling -Initialization_cycle On), which is the default, using the -Noinitialization switch when saving patterns may result in simulation mismatches when you verify the patterns in a timing-based simulator.

- -BEgin {*pattern_number* | *pattern_name*}

  An optional switch that specifies the pattern to begin storing test patterns. Usually used to specify the first pattern in a range to include in the test patterns. The default pattern is 0. For Tessent FastScan and Tessent TestKompress, *pattern_number* is an integer and *pattern_name* is a string generated by using the -tag switch (which specifies a prefix for all pattern names). For example, *pattern_name* = *tag_name*_1, *tag_name*_2, and so on. When used together with "-pattern_sets chain" in the same command, the -Begin switch applies to chain test patterns only. When used together with -Scan in the same command, the -Begin switch applies to scan patterns only. When used together with -All in the same command, the -Begin switch applies to scan patterns only and writes out all chain patterns.

  Only one -Begin argument can be used per write_patterns command. To save multiple ranges of patterns, for example patterns 11 through 20 and 81 through 100, use the

set_pattern_filtering command with the -Range and/or -Norange switches before you issue the write_patterns command.

If you save only a portion of the internal patterns in the -Ascii format, test coverage and fault information is not included.

- -END {*pattern_number* | *pattern_name*}

  An optional switch that specifies the last pattern of a range to include in the test patterns. This argument is inclusive; therefore, the tool stores the pattern identified by the *pattern_number* or *pattern_name* you specify. The default pattern is the last pattern of the pattern set. For Tessent FastScan and Tessent TestKompress, *pattern_number* is an integer and *pattern_name* is a string generated by using the -tag switch (which specifies a prefix for all pattern names). For example, *pattern_name* = *tag_name*_1, *tag_name*_2, and so on. When used together with "-pattern_sets chain" in the same command, the -End switch applies to chain test patterns only. When used together with -Scan in the same command, the -End switch applies to scan patterns only. When used together with -All in the same command, the -End switch applies to scan patterns only and writes out all chain patterns.

  If you save only a portion of the internal patterns in the -Ascii format, the tool does not include test coverage and fault information.

- -TAg *tag_name*

  An optional switch that adds a unique user-specified label, *tag_name*, to each pattern. All chain tests automatically have "chain as the *tag_name*" regardless of the tag_name given in the -Tag switch. Since all tag names must be unique, "chain" is not a valid *tag_name*. If the *tag_name* supplied is not unique, an error message is issued and the run aborts.

  If the tool reads in patterns using the Set Pattern External command, the patterns must also be unique. The run aborts if the tool attempts to make two identically named patterns co-resident by any method. This uniqueness extends across both the internal and external pattern sets. It is not possible to have the same pattern_name in the internal and external sets.

- -CEll_placement Bottom | Top | None

  An optional switch and literal pair that controls the placement of the scan cell data in the ASCII pattern file. The literal choices are as follows.

  Bottom — A literal that places the scan data after the patterns, at the end of the file. This is the default.

  Top — A literal that places the scan data before the patterns, at the top of the file.

  None — A literal that excludes the scan data from the file.

- -ENVironment

  An optional switch that adds information about the current Tessent FastScan or Tessent TestKompress environment to the pattern file as comments. The information includes the identification stamp number, the environment settings, and the DRC rules. The information is the same as the report_environment and report_id_stamp commands display.

- -TEST_Setup { ONE_Per_file | OFf | ONLy | ONE }

  An optional switch and literal pair that specifies how the command saves the test_setup procedure. If you do not use this switch, the command writes one test_setup procedure at the start of each pattern file.

  The literal choices are as follows.

  ONE_Per_file — A literal that places one test_setup procedure at the start of each pattern file, even when multiple pattern files are saved with one command using the -max_loads, -memory_size, or -scan_memory_size switches. This is the default option.

  OFf — A literal that saves patterns without including the test_setup vectors. For ASCII and Binary patterns, this option has the same effect as the “One” option. Using this option infers the use of the -NOInitialization switch as well. This is because the init vector is not needed in the pattern file when the test_setup vectors are not present.

  Note that using this option might produce vectors that do not simulate correctly or might produce failures if used on a tester.

  ONLy — A literal that saves the test_setup procedure as a separate file. This option does not work if you specify any of the following switches: “-test_end only,” -pattern_sets, -begin, -end.

  ONE — A literal that applies only one test_setup procedure when both chain and scan test patterns are saved in one pattern file. When saving patterns in formats other than ASCII and binary, a single test_setup procedure is saved in the first file generated when the -Maxloads switch is used with this “One” option.

- -TEST_End { ONE | ONLy | OFf }

  An optional switch and literal pair that specifies how the command saves the test_end procedure. If you don’t use this switch, one test_end procedure is applied at the end of the pattern file. The literal choices are as follows.

  ONE — A literal that applies one test_end procedure at the end of the pattern file after all pattern sets are done. This is the default option. When you also use -Maxloads, then the tool saves a test_end procedure at the end of each generated pattern file.

  ONLy — A literal that saves the test_end procedure as a separate file. This option does not work if you specify any of the following switches: “-test_setup only,” -pattern_sets, -begin, -end.

  OFf — A literal that saves patterns without including the test_end vectors. For ASCII and Binary patterns, this switch has no effect. Note that using this switch might produce vectors that don’t simulate correctly or might produce failures if used on a tester.

- -EXternal

  An optional switch that saves the current external pattern set to the specified *pattern_filename*. By default, internal test patterns are saved.

  External patterns saved in ASCII format do not include test coverage and fault information.

- -SAMple [*integer*]

> **Tip**: This switch has been superseded by the set_pattern_filtering command, which has multiple sampling options.

An optional switch that specifies to save a sample of patterns for each pattern type instead of saving all the patterns. This can be useful for serial pattern simulation to shorten the simulation time. In Tessent FastScan, the different types are those reported by the report_statistics command. In Tessent TestKompress, the types are additionally based on whether or not the tool used masking in a pattern. For example, in Tessent TestKompress, a clock-sequential pattern with scan chain masking is considered a different type than a clock-sequential pattern without scan chain masking.

The optional *integer* specifies the number of patterns contained in each sample. If you do not include an *integer*, the default is to sample one pattern per type.

This switch is not supported in the patterns -scan_retargeting context.

- -IDDQ_file

  An optional switch that creates an ASCII file that lists the cycle number, vector number, and measure time for each saved pattern that has an IDDQ measurement point. This IDDQ information file has the same name as the pattern file, with ".IDDQ" appended to the end and is created for all enhanced pattern output formats. This switch does not support ASCII or binary output formats.

- -MODE_Internal

  An optional switch that saves test patterns using the internal mode of the named capture procedures. Test patterns saved with this switch treat all internal clocks and internal primary inputs as directly controllable pins. This is the default when you write patterns in ASCII or binary format.

- -MODE_External

  An optional switch that specifies to write patterns using the external mode of the named capture procedures and no user-defined internal pins. When setting up for ATPG, if you used the add_clocks -Internal or add_primary_inputs -Internal command to define one or more internal pins, then you need to use this switch when saving patterns. This is the default when you write patterns in tester formats (for example, WGL) and Verilog format.

> **Note**
> If a capture procedure does not contain an external mode, the -Mode_external switch uses the internal mode cycle information, but ignores internal PI events. The switch also ignores internal pin events in scan procedures (shift, load_unload, and so on).

- -Lib *work_dir*

  An optional switch and string pair that specifies the working directory containing the compiled top level design. The default *work_dir* is "work" in the current directory.

- -NAme_match

  An optional switch that instantiates the device under test in the test bench using the name matching method. If you start with a Verilog netlist and do not restart with a flattened model, the Verilog testbench uses the position matching method.

- -POSition_match

  An optional switch that places the device under test in the test bench using the position matching method.

- -NOPadding

  An optional switch that removes X values from the input data in saved ASCII test patterns. By default, X values are used for unknown or *don't care* bits in the input data. Use this switch to eliminate X values in the input data due to short scan chains. This switch can only be used for ASCII test patterns.

  To load unpadded test patterns into a tool, use the read_patterns command.

- -PAD0 | -PAD1

  For EDT Off:

  Optional switches that replace X values with either 1 or 0 value in the saved test patterns. By default, X values are used for unknown or *don't care* bits in the input and output data. Options include:

  -PAD0 — Replaces X values with 0 values.

  -PAD1 — Replaces X values with 1 values.

  This switch is only valid for test patterns that support scan chain padding.

  For EDT On:

  Optional switches that replace X values with binary values in the saved test patterns. By default, X values are used for unknown and *don't care* bits in the output data. This switch only affects the unload values in the test patterns. The binary value may be a 0 or 1 and cannot be specified. Both -PAD0 and -PAD1 operate the same on Tessent TestKompress test patterns.

- -Noz

  An optional switch that changes all Z output values to the value specified by the last set_z_handling command.

- -MAxloads *load_number*

  An optional switch and positive integer pair that specifies the number of scan loads, *load_number*, to include in a pattern file. Much like the -Begin and -End switches, the -Maxloads switch automatically breaks up the pattern file into a series of pattern files with the specified number of scan loads in each. For example, suppose pattern creation results in 730 basic scan patterns. The following command that uses -Maxloads,

```
write_patterns my_patterns -maxloads 100
```

saves the same sequence of files with the same pattern content as this series of commands:

```
write_patterns my_patterns_1 -begin 0 -end 99
write_patterns my_patterns_2 -begin 100 -end 199
...
write_patterns my_patterns_7 -begin 600 -end 699
write_patterns my_patterns_8 -begin 700
```

In either case, every file except the last one would contain 100 patterns; the last file would contain 30 patterns. You can also use -Maxloads to break up a pattern subset you specify with the -Begin and/or -End arguments in the same command. For example, here is the command to write patterns 50 through 99 to a sequence of files that each contain no more than 15 patterns:

```
write_patterns my_patterns -maxloads 15 -begin 50 -end 99
```

When you use -Maxloads, the tool automatically appends an underscore and sequence number to the name of each resultant pattern file, starting with number one (_1).

Note

This switch is only valid with ASCII, Binary, and enhanced pattern output and cannot be used with the -Pattern_size switch.

- -PATtern_size *integer*

  A optional switch and integer pair which specifies the size of the memory buffer and pattern file in which to save. *Integer* is given in kilobytes.

Note

The -Pattern_size switch is valid only when using -Verilog output.

The default pattern size is 128MB. However, any size specified for a pattern size will be adjusted to hold a multiple of the largest pattern. For example, if the largest Tessent FastScan or Tessent TestKompress pattern requires 3MB for one pattern, then the file size will be a multiple of 3MB, which would result in a 126MB pattern size.

- -MEMory_size *size_in_KB*

  An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of available main tester memory, and sets this number as a limit on the size of the pattern data in a pattern file. Much like the -Maxloads switch, the -Memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes (KBs) of pattern data. The tool assumes that one pattern value (0, 1, X, or Z) requires one byte on the tester.

  If you specify this switch and the -Scan_memory_size switch, the -Memory_size switch specifies the KB limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

Note that the number of vectors in each pattern file will vary. This is because each file contains the vectors from the test_setup procedure and all the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

- -SCAN_Memory_size *size_in_KB*

  An optional switch and integer pair that specifies the maximum size, *size_in_KB*, of tester memory available for only scan pattern data, and sets this number as a limit on the size of this data in the pattern file. Much like the -Maxloads switch, the -Scan_memory_size switch automatically breaks up a large pattern file into a series of smaller pattern files that each contain no more than the specified number of kilobytes (KBs) of scan pattern data. The tool assumes one scan pattern value (0, 1, X, or Z) requires one byte on the tester.

  If you specify this switch and the -Memory_size switch, the latter specifies the KB limit for only the non-scan pattern data. In this case, the tool ends each pattern file when the size of either the scan data or the non-scan data reaches its limit.

Note

You cannot use the -Scan_memory_size switch and the -Serial switch in the same command.

Note that the number of vectors in each pattern file varies; this difference occurs because each file contains the vectors from the test_setup procedure and the vectors for each Tessent FastScan pattern it contains. Vectors for a single pattern are not split between pattern files.

- -VERBose

  An optional switch that writes extra information messages about the files being saved to stdout or the tool's log file. These messages use the following syntax:

  ```
  // Writing file_type file_name
  ```

  Where *file_type* is a phrase describing what the file is, and *file_name* is the specified or generated file name. For example, *file_type* could be "STIL patterns," "WGL patterns," "Verilog testbench," "Verilog vectors," and so forth.

- [{-EDT_Bypass | - EDT_Single_bypass_chain} | -EDT_Internal | -EDT_External] **((EDT On)**

  Optional switch that specifies how EDT test patterns are saved. By default, Tessent TestKompress (EDT on) saves ASCII test patterns that define load data with respect to the input channels of the EDT logic and unload data with respect to the core scan chains. Options include:

  -EDT_Bypass — An optional switch used with the -BInary format switch to save EDT bypass test patterns in binary format. EDT bypass test patterns are the decompressed version of the EDT test patterns. These options are only valid for EDT test pattern generation. For more information, see "Generation of Identical EDT and Bypass Test Patterns" in the *Tessent TestKompress User's Manual*.

Note

This option cannot be used when bypass chains are inserted in the design netlist before the EDT logic is generated.

-EDT_Single_bypass_chain — An optional switch used with the -BInary format switch to save EDT bypass test patterns for the single bypass chain configuration. The test patterns are saved in binary format. These options are only valid for EDT test pattern generation. For more information, see "Dual Bypass Configurations" in the *Tessent TestKompress User's Manual*.

-EDT_Internal — An optional switch that saves EDT test pattern data for only the internal core scan chains to an ASCII file. The test patterns contain core scan chain force and observe data except for X values observed on the output of the compactor due to X blocking or scan chain masking. For more information, see "Understanding Scan Chain Masking in the Compactor" and "EDT Internal Patterns" in the *Tessent TestKompress User's Manual*.

Note

Test patterns saved with the -Edt_internal switch cannot be read back into the tool.

-EDT_External — An optional switch that saves EDT test pattern data with respect to the (compressed) input and output channels. These test patterns are similar to WGL and STIL tester formats and must be resimulated before they can be saved out to an external test pattern file.

- ***filename* (Hybrid TK/LBIST Flow Only)**

  A required string that specifies the name of the file to write the register values to.

- -replace **(Hybrid TK/LBIST Flow Only)**

  An optional switch that directs the tool to overwrite an existing ***filename***.

- -patdb **(Hybrid TK/LBIST Flow Only)**

  An optional switch that directs the tool to write out a PatternDB file.

Example 1

The following example performs an ATPG run, and then saves only the first 15 test patterns to a file in the Verilog format, including the timing information contained in the timing file:

```
set_system_mode analysis
create_patterns
write_patterns file1 -verilog timefile -end 14
```

Example 2

The following example illustrates the module name created in the enhanced Verilog output when using the -Procfile switch. If the design name is MAIN, and you issue the following write_patterns command:

```
write_patterns pattern1.pat -procfile -verilog -replace
```

the module name in the testbench will be MAIN_pattern1_pat_ctl.

Example 3

The following example (in patterns -scan) saves Verilog chain test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the chain test patterns (for debugging scan chain problems). The example assumes the parameter file, test.param, contains a “SIM_SHIFT_DEBUG 1;” line.

**write_patterns chain_shift_debug_patt.v -verilog -pattern_sets chain -serial \
-parameter test.param**

Example 4

The following examples illustrate the output of the -Verbose switch:

write_patterns test_pat.v –verilog –replace -verbose

```
// Writing Verilog testbench test_pat.v
// Writing Verilog PO names test_pat.v.po.name
// Writing Verilog chain names test_pat.v.chain.name
// Writing Verilog vectors test_pat.v.0.vec
// Writing Verilog vectors test_pat.v.1.vec
```

write_patterns test_pat.stil –stil –replace –verbose –maxloads 10

```
// Writing STIL patterns test_pat_0.stil
// Writing STIL patterns test_pat_1.stil
```

Example 5

The following example writes the test_setup procedure as a separate STIL file:

write_patterns test_setup.stil -stil -replace -test_setup only

Related Commands

add_scan_groups
analyze_simulation_mismatches
read_patterns
read_procfile
report_measure_cycles
report_procedures
report_timeplate
set_external_simulator
set_pattern_classification
write_procfile

write_primary_inputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

write_primary_inputs ***filename*** [-Replace] [-All | *primary_input_pin*…]

Description

Writes primary inputs to the specified file.

The write_primary_inputs command writes a list of either all the primary inputs of a circuit or a specific list of primary inputs that you specify into a file where it can be reviewed.

This command is identical to the report_primary_inputs command except the data is written into a file.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool writes the primary inputs.

- -Replace

  An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

- -All

  An optional switch that specifies to write all the primary inputs to the file. This is the default.

- *primary_input_pin*

  An optional repeatable string that specifies a list of primary input pins that you want to write to the file.

Examples

The following example writes all primary inputs to a file:

```
write_primary_inputs inputfile
```

Related Commands

report_primary_inputs

write_primary_outputs

Context: dft -scan, dft -test_points

Mode: setup, analysis

Usage

write_primary_outputs ***filename*** [-Replace] [-All | *primary_output_pin*…]

Description

Writes primary outputs to the specified file.

The write_primary_outputs command writes a list of either all the primary outputs of a circuit or a specific list of primary outputs that you specify into a file where it can be reviewed.

This command is identical to the report_primary_outputs command except the data is written into a file.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool writes the primary outputs.

- -Replace

  An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

- -All

  An optional switch that specifies to write all the primary outputs to the file. This is the default.

- *primary_input_pin*

  An optional repeatable string that specifies a list of primary output pins that you want to write to the file.

Examples

The following example writes all primary outputs to a file:

```
write_primary_outputs outputfile
```

Related Commands

report_primary_outputs

write_procedure_testbench

Context: dft -edt, patterns -scan

Mode: analysis

Usage

write_procedure_testbench ***testbench_filename*** [-Replace]
[-PARAMeter *parameter_filename*] [-ALl | -CApture *capture_name*… | -LOad_unload]

Description

Writes a test bench for verifying correct operation of internal signals used in procedures.

The write_procedure_testbench command writes a Verilog testbench for verifying correct operation of internal signals used in procedures, including named capture procedures. The testbench will verify that if a procedure forces or pulses an internal signal, the value forced or pulsed is actually produced by the circuit when simulated in a Verilog simulator. (The expect values in the testbench are based on the statements in the procedure file that force or pulse internal pins.)

Note

The testbench written by this command is unrelated to the testbench written by the write_patterns command, which enables you to simulate and verify the ATPG patterns. Saving patterns is the last step of the ATPG process, whereas you should write a testbench to verify named capture procedures before you create patterns.

Determining from incorrect patterns where the mistakes are in the procedures can be difficult. This command helps you find the mistakes in the procedures, so you can correct them *before* they result in incorrect patterns.

Arguments

- ***testbench_filename***

  A required string that specifies the name of the file to which you want to write the testbench.

- -Replace

  An optional switch that replaces the contents of *testbench_filename* if the file already exists. By default, the command will not overwrite an existing file.

- -PARAMeter *parameter_filename*

  An optional switch and string pair that specifies an external parameter file to retrieve field names for use in certain output formats. For more information on the format of the parameter file, refer to the "Parameter File Format and Keywords" on page 2679.

- -All

  An optional switch that specifies to write a testbench that simulates the test_setup procedure, followed by all named capture procedures simulated as load_unload/capture pairs. Values forced on internal signals in all procedures are used as expected responses.

Named capture procedures that do not force values on internal signals are skipped. An error message is issued if none of the procedures force values on internal signals. This is the default.

- -CAPture *capture_name*

  An optional switch and repeatable string that specifies a list of named capture procedures to verify. In this case, the command will write a testbench that simulates the test_setup procedure, followed by the load_unload procedure with shift procedure, and then the specified named capture procedure(s). If you specify more than one named capture procedure, the testbench simulates the additional ones as additional load_unload/capture pairs. Each load_unload/capture pair is simulated twice. Values forced on internal signals in the named capture procedures are used as expected responses. An error message is issued if a specified named capture procedure does not force any values on internal signals.

- -LOAd_unload

  An optional switch that specifies to write a testbench that simulates the load_unload and shift procedures. This testbench simulates the test_setup procedure (if present), followed by the load_unload procedure with one shift. Values forced on internal signals in the load_unload and shift procedures are used as expected responses.

Example 1

The following example writes a Verilog testbench for verifying the functionality of the named capture procedure “cap1.”

write_procedure_testbench verilog_tb_cap1.v -capture cap1 -replace

Example 2

The following example assumes the pathname of the Model*Sim* bin directory has previously been added to the PATH variable. From the shell, the example first creates a Verilog library named “proc_test_work,” then compiles the following three elements into the library:

- source for the top level design
- testbench written out in the preceding example
- parts library

With these prerequisites met, the example simulates the testbench.

```
vlib proc_test_work
vlog -work proc_test_work counter_syn_scan.v verilog_tb_cap1.v \
   -v my_verilog_source/verilog_lib.v
Model Technology ModelSim SE vlog 6.1 Compiler 2005.06 Jun  6 2005
-- Compiling module mux21_macro
-- Compiling module sffr_macro
-- Compiling module lmmux
...
-- Scanning library file ’my_verilog_source/verilog_lib.v
-- Compiling UDP dff_r_err
-- Compiling UDP dff_r
```

```
Top level modules:
        counter_verilog_tb_cap1_v_ctl

vsim counter_verilog_tb_cap1_v_ctl -lib proc_test_work \
   -t 100ps -c -do "run -all"
...
# //
# Loading proc_test_work.counter_verilog_tb_cap1_v_ctl
# Loading proc_test_work.counter
# Loading proc_test_work.count4
...
# Loading proc_test_work.dff_r
# Loading proc_test_work.dff_err
# Loading proc_test_work.dff_p
# run -all
#         1850: inst_0/QI_nx7 simulated 0  expected 1
#         1850: Procedure: CAPTURE1 Event: force inst_0/QI_nx7 to 1 at 0
```

The last two lines indicate the procedure named “capture1” forces a 1 on the internal pin /inst_0/QI_nx7, but the value produced on the pin when simulated in the Verilog simulator is 0. This suggests the force statement is applying an incorrect value. The next example shows what the corresponding force statement (highlighted in bold) might look like in the test procedure file:

```
procedure capture capture1 =
   scan_group groupcnt ;
   mode internal =
      timeplate gen_tp1 ;
      cycle =
         force clear 0 ;
         force clock[0] 0 ;
         ...
         force clock[9] 0 ;
         force test_clk 0 ;
         force /inst_0/QI_nx7 1;
         force_pi ;
         measure_po ;
         pulse clock[0] ;
         ...
         pulse clock[4] ;
      end;
   end;
   mode external =
      timeplate gen_tp1_long ;
      cycle =
         force clear 0 ;
         force clock[0] 0 ;
         ...
         force clock[9] 0 ;
         force test_clk 0 ;
         force_pi ;
         measure_po ;
         pulse clock[0] ;
         ...
         pulse clock[4] ;
      end;
   end;
```

```
end; // capture1
```

Example 3

The following example shows what the message at the end of the Verilog simulation run looks like if there are no mismatches:

```
# run -all
# No error between simulated and expected patterns.
```

Related Commands

report_capture_procedures
report_procedures
report_timeplate
set_capture_procedures

write_procfile

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

Mode: setup, analysis

Usage

write_procfile ***proc_filename*** [-Replace] [-Full [-EXternal]]

Description

Writes existing procedure and timing data to the named test procedure file.

The write_procfile command writes out existing procedure and timing data to the named test procedure file.

Arguments

- ***proc_filename***

  A required string that specifies the name of the file to which you want to write existing procedure and timing data.

- -Replace

  An optional switch that replaces the contents of the file if the *proc_filename* already exists.

- -Full

  An optional switch that causes the tool to parse the ATPG pattern list (if any) and create all needed non-scan procedures before writing the procedure file data. In Tessent FastScan and Tessent TestKompress, this option will always create a capture procedure.

Note

The -Full option can cause the write_procfile command to take more time if there are a large number of ATPG-generated patterns in the internal pattern list.

- -EXternal

  An optional switch that causes the -Full switch to use the external patterns specified by the read_patterns command instead of internally generated patterns. This switch behavior is similar to the -External switch behavior for the write_patterns command. The -External switch is only valid when using the -Full switch.

Example

The following example writes the existing procedure and timing data to the specified file:

```
write_procfile myprocfile.proc
```

Related Commands

add_scan_groups
read_procfile
report_procedures
report_timeplate
write_patterns

write_scan_order

Context: dft -scan

Mode: all modes

Usage

write_scan_order ***filename*** [-Replace] [-No_segmentation] [-Keep_short_segments]

Description

Writes out the scan chains in the design using the DEF language.

The write_scan_order command writes the chains into the SCANCHAINS section of the specified DEF file, where each statement describes a single scan chain. When the tool successfully traces through a subchain, the cells that comprise that subchain are written to the FLOATING list of the corresponding scan chain's statement by default. In this case, the order of cells comprising such a subchain doesn't have to be preserved, and cells can appear in the START and STOP sections.

You can override the default behavior when adding subchains with the add_sub_chains command by using the -No_reordering switch. In this case, the tool preserves the order of cells that comprise the subchain by writing them to the ORDERED list of the corresponding scan chain's statement. The tool tries to avoid having any subchain cells at scan chain boundaries during scan chain stitching in order to have all these cells in the same ORDERED list (that is, none of them appear in the START and STOP sections). If there is no scan cell available that is not part of a subchain, the tool inserts lockup cells at the boundaries of the scan chains.

Each shift register is written into a separate ORDERED list of the corresponding scan chain's statement.

Arguments

- ***filename***

  A required string that specifies the name of the DEF file.

- **-Replace**

  An optional switch that replaces the specified DEF file, if it exists.

- **-No_segmentation**

  An optional switch that disables the segmentation of scan chains in the DEF file. By default, scan chains with different clocks/edges are segmented into separate chains.

  If you disable the segmentation of scan chains, cells of scan chains with different clock/edge domains are written out into a single ORDERED list and the PARTITION and MAXBITS information is not written out for these scan chains in the DEF file. This prevents the reordering of scan flip-flops along the scan chains and across the scan chains.

- -Keep_short_segments

  An optional switch that specifies that scan chains that contain only one or two scan cells and contain no lockup latches are written to the scan DEF file and are not commented out; by default, scan chains of this type are written to the scan DEF file and are commented out.

Example 1

This example illustrates the information reported with the report_scan_cells command and written to the scanDEF file.

report_scan_cells

```
-------------------------------------------------------------------------
        Chain   Group            ShiftReg   Library             Clock
Clock
CellNo  Name    Name   Pathname  ID/CellNo  ModelName  ScanOut  Pinname
Polarity
-------------------------------------------------------------------------
0      chain1  dummy  /ud5     -/-       sff       QB      clk1    (+)
1      chain1  dummy  /ud4     1/4       dff       QB      clk1    (+)
2      chain1  dummy  /ud3     1/3       dff       Q       clk1    (+)
3      chain1  dummy  /ud2     1/2       dff       Q       clk1    (+)
4      chain1  dummy  /ud1     1/1       sff       Q       clk1    (+)
5      chain1  dummy  /ud10    3/2       dff       Q       clk1    (+)
6      chain1  dummy  /ud9     3/1       sff       Q       clk1    (+)
7      chain1  dummy  /ud8     -/-       sff       QB      clk1    (+)
0      chain2  dummy  /ud11    -/-       sff       Q       clk1    (+)
1      chain2  dummy  /ud7     2/2       dff       QB      clk1    (+)
2      chain2  dummy  /ud6     2/1       sff       Q       clk1    (+)
3      chain2  dummy  /ud12    -/-       sff       Q       clk1    (+)
4      chain2  dummy  /ud13    -/-       sff       Q       clk1    (+)
5      chain2  dummy  /ud14    -/-       sff       Q       clk1    (+)
6      chain2  dummy  /ud15    -/-       sff       Q       clk1    (+)
-------------------------------------------------------------------------
```

The tool identifies three shift registers as shown in the following output generated by the report_shift_registers command.

report_shift_registers -verb

```
-------------------------------------------------------------------
             Hierarchical  SequentialCell  Clock        Library
Id   Length  Path          InstanceName    Edge & Name  ModelName
-------------------------------------------------------------------
[1]  4       /             ud1             + clk1       sff
                           ud2             + clk1       dff
                           ud3             + clk1       dff
                           ud4             + clk1       dff
[2]  2       /             ud6             + clk1       sff
                           ud7             + clk1       dff
[3]  2       /             ud9             + clk1       sff
```

Given this input, the following command generates the scan DEF file that follows.

write_scan_order results/scandef -replace

Note that in the following file, regular flip-flops and shift register flip-flops within the same domain are separated by FLOATING and ORDERED lists, respectively. Each shift register is written out in a separate ORDERED list.

```
#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 2 ;

- chain1_sub0
  + START ud8 QB
  + ORDERED
      ud9 ( IN SI ) ( OUT Q )
      ud10 ( IN D ) ( OUT Q )
  + ORDERED
      ud1 ( IN SI ) ( OUT Q )
      ud2 ( IN D ) ( OUT Q )
      ud3 ( IN D ) ( OUT Q )
      ud4 ( IN D ) ( OUT QB )
  + STOP ud5 SI

  # Partition for core chain in clock clk1 (pos-edge) domain
  + PARTITION partition_1  MAXBITS 6 ;

- chain2_sub0
  + START ud15 Q
  + FLOATING
      ud14 ( IN SI ) ( OUT Q )
      ud13 ( IN SI ) ( OUT Q )
      ud12 ( IN SI ) ( OUT Q )
  + ORDERED
      ud6 ( IN SI ) ( OUT Q )
      ud7 ( IN D ) ( OUT QB )
  + STOP ud11 SI

  # Partition for core chain in clock clk1 (pos-edge) domain
  + PARTITION partition_1  MAXBITS 5 ;

END SCANCHAINS

END DESIGN
```

Similarly, given the same input, the following command generates the scan DEF file that follows:

write_scan_order –no_segmentation scan.def

Note the following about the scan DEF file:

- sub1 and sub2 are instances of library cell libsubchmodel, which contains a subchain of length 2. The scan DEF includes this length in the BITS field. For regular cells of length 1, the BITS field is not printed.
- All flip-flops are written out in one ORDERED list because chain1 contains cells in different clock edge domains.

```
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN A ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1
  + START u5 Q
  + ORDERED
      u1 ( IN SI ) ( OUT Q )
      u2 ( IN D ) ( OUT Q )
      u3 ( IN D ) ( OUT Q )
      u4 ( IN D ) ( OUT Q )
      u6 ( IN SI ) ( OUT Q )
      u7 ( IN D ) ( OUT Q )
      u8 ( IN SI ) ( OUT Q )
      u9 ( IN D ) ( OUT Q )
      u10 ( IN D ) ( OUT Q )
      u11 ( IN SI ) ( OUT Q )
      u12 ( IN D ) ( OUT Q )
      u13 ( IN SI ) ( OUT Q )
      u14 ( IN D ) ( OUT Q )
      u15 ( IN SI ) ( OUT Q )
      lckup1 ( IN D ) ( OUT Q )
      sub1 ( IN sci1 ) ( OUT sco1 ) ( BITS 2 )
  + STOP sub2 sci2 ;

END SCANCHAINS
```

Example 2

This example illustrates the information reported with the report_scan_cells command and written to the scanDEF file when traceable subchains are present in the design.

report_scan_cells

```
-------------------------------------------------------------------------------
       Chain            Group                                                Clock
CellNo Name             Name   Pathname         CellName  ScanOut   Clock  Polarity
-------------------------------------------------------------------------------
-      chain1           dummy  /lckup1          latch     Q         clk2   (-)
0      chain1 (schi2)   dummy  /uB/f3           sff       Q         clk2   (+)
1      chain1 (schi2)   dummy  /uB/f2           sff       Q         clk2   (+)
2      chain1 (schi2)   dummy  /uB/f1           sff       Q         clk2   (+)
-      chain1           dummy  /lckup2          latch     Q         clk2   (+)
0      chain2 (schc1)   dummy  /uWA/uA/f21      sff       Q         clk    (+)
1      chain2           dummy  /ud              sff       Q         clk    (+)
```

```
2      chain2           dummy  /us              sff       Q         clk    (+)
0      chain3 (scho1)   dummy  /uWA/uA/f31      sff       Q         clk    (+)
-      chain4           dummy  /lckup3          latch     Q         clk    (-)
0      chain4 (schi1)   dummy  /uWA/uA/f3       sff       Q         clk    (+)
1      chain4 (schi1)   dummy  /uWA/uA/bb/f2    sff       Q         clk    (+)
2      chain4 (schi1)   dummy  /uWA/uA/f1       sff       Q         clk    (+)
-      chain4           dummy  /lckup4          latch     Q         clk    (+)
```

write_scan_order

```
#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1_sub0
  + START lckup2 Q
  + FLOATING
      uB/f1 ( IN SI ) ( OUT Q )
      uB/f2 ( IN SI ) ( OUT Q )
      uB/f3 ( IN SI ) ( OUT Q )
  + STOP lckup1 D

  # Partition for core chain in clock clk2 (pos-edge) domain
  + PARTITION partition_1  MAXBITS 3 ;

- chain2_sub0
  + START us Q
  + FLOATING
      ud ( IN SI ) ( OUT Q )
  + STOP uWA/uA/f21 SI

  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 1 ;
#  The following chain segment with only 1 or 2 scan cells has been
#  commented out for compatibility with the layout tools.
#- chain3_sub0
#  + START uWA/uA/f31 SI
#  + STOP uWA/uA/f31 Q ;
- chain4_sub0
  + START lckup4 Q
  + FLOATING
      uWA/mux ( IN A1 ) ( OUT Y ) ( BITS 0 )
      uWA/uA/f1 ( IN SI ) ( OUT Q )
      uWA/uA/bb/f2 ( IN SI ) ( OUT Q )
      uWA/uA/f3 ( IN SI ) ( OUT Q )
  + STOP lckup3 D

  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 3 ;

END SCANCHAINS

END DESIGN
```

Example 3

The following example is a modification of previous example. In this example, the set_attribute_value command assigns the attribute is_hard_module to the module blackBox. An instance of this module, uWA/uA/bb, is written to the scanDEF file unexpanded and the number of scan cells on its scan path appears in the BITS statement. Additionally, the -no_reordering switch was used with the add_sub_chains command when defining subchains; therefore, the flip-flops contained within the subchains are written to the ORDERED lists.

set_attribute_value blackBox -name is_hard_module -value true

```
#
VERSION 5.7 ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;
UNITS DISTANCE MICRONS 1000 ;

SCANCHAINS 1 ;

- chain1_sub0
  + START lckup2 Q
  + ORDERED
      uB/f1 ( IN SI ) ( OUT Q )
      uB/f2 ( IN SI ) ( OUT Q )
      uB/f3 ( IN SI ) ( OUT Q )
  + STOP lckup1 D

  # Partition for core chain in clock clk2 (pos-edge) domain
  + PARTITION partition_1  MAXBITS 3 ;

- chain2_sub0
  + START us Q
  + FLOATING
      ud ( IN SI ) ( OUT Q )
  + STOP uWA/uA/f21 SI

  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 1 ;


#  The following chain segment with only 1 or 2 scan cells has been
#  commented out for compatibility with the layout tools.
#  - chain3_sub0
#  + START uWA/uA/f31 SI
#  + STOP uWA/uA/f31 Q ;

- chain4_sub0
  + START lckup4 Q
  + ORDERED
      uWA/mux ( IN A1 ) ( OUT Y ) ( BITS 0 )
      uWA/uA/f1 ( IN SI ) ( OUT Q )
      uWA/uA/bb ( IN si ) ( OUT so ) ( BITS 1 )
      uWA/uA/f3 ( IN SI ) ( OUT Q )
  + STOP lckup3 D
```

```
  # Partition for core chain in clock clk (pos-edge) domain
  + PARTITION partition_2  MAXBITS 3 ;

END SCANCHAINS

END DESIGN
```

Example 4

The following example shows command output when power domains are present in the design:

write_scan_order

```
- chain1_sub0
  + START I2/df_0003_1 QB
  + FLOATING
      I2/df_0002_1 ( IN SI ) ( OUT QB )
  + STOP I2/df_0001_1 SI
  # Partition for core chain in clk (pos-edge) clock domain of
  # PD2 power domain
  + PARTITION partition_1  MAXBITS 1 ;


- chain2_sub0
  + START I10/df_0002_1 QB
  + FLOATING
      I10/df_0001_1 ( IN SI ) ( OUT QB )
  + STOP I10/df_0003_1 SI
  # Partition for core chain in clk (pos-edge) clock domain of
  # PD3 power domain
  + PARTITION partition_2  MAXBITS 1 ;
```

Related Commands

add_scan_pins

add_sub_chains

write_scan_setup

Context: dft -test_points

Mode: insertion

Usage

write_scan_setup **-prefix *filename_prefix*** [-replace] [-all_internal_clocks]

Description

Writes out a dofile containing all necessary steps for scan insertion.

The existing chain and sub-chain information will be carried over from previous dofiles.

Arguments

- **-prefix *filename_prefix***

  A required switch and string that specifies a name prefix for the scan setup files.

  The filenames are composed of this prefix followed by an underscore (_) and a descriptive file-specific suffix. You can include the pathname to an existing directory as a component of the ***filename_prefix***, and the files are written to that location.

- -replace

  An optional switch that replaces the contents of each file that already exists.

- -all_internal_clocks

  An optional switch that specifies to write all internally defined clocks to the timeplate, to the ATPG dofile, and to the shift procedure.

Example

The following example writes out a dofile and test procedure file that can be used from the dft -scan context to perform x-bounding and scan insertion on the design:

```
analyze_test_points
insert_test_logic
write_design -output netlist_with_test_points.v -replace
write_scan_setup -prefix scan_setup -replace
```

Related Commands

analyze_test_points

insert_test_logic

write_subchain_setup

Context: dft -scan

Mode: analysis, insertion

Usage

write_subchain_setup ***filename*** -Replace

Description

Writes the appropriate add_sub_chains commands to a file so that the tool can understand the pre-existing scan subchains at the top-level of this module.

The write_subchain_setup command is useful if you are performing a block-by-block test synthesis at the lower design level. By using this command to specify for the tool to write the subchain setup information at the lower-level and then reading in the setup file as part of the scan subchain setup, you can avoid having to define the pre-existing scan subchain at the higher design level with the add_sub_chains command.

When the *-channels* switch of the set_edt_options command is used, subchain declarations are written as if the block was also inserted into EDT logic and the scan chains were concatenated into EDT bypass chains. For more information, see the description for the set_edt_options command.

Additionally, if the set_wrapper_chains -Wrapper_extest_ports switch is used, a second group of subchain declarations is written out for the wrapper chains accessed through the new extest ports. For more information, see the description for the set_wrapper_chains command.

Arguments

- ***filename***

  A required string that specifies the name of the file to which the tool writes the scan subchain setup information. For information on this file format, refer to the insert_test_logic command.

- -Replace

  An optional switch that specifies for the tool to replace the contents of the file, if the file already exists.

Examples

The following example writes the scan chain setup information for this module so that you can later use the *module1.setup* file at the top-level to define the pre-existing scan subchain:

```
insert_test_logic
write_subchain_setup /user/designs/module1.setup
```

The following example shows the contents of the *module1.setup* file:

```
add_sub_chains /user/designs/sub1 subc1 scan_in1 scan_out1 7
        mux_scan scan_en
```

Related Commands

add_sub_chains
insert_test_logic
report_sub_chains
set_edt_options
set_wrapper_chains

write_test_point_dofile

Context: dft -test_points

Mode: all modes

Usage

write_test_point_dofile **-output_file *filename*** [-all | -control | -observe] [-tool_analyzed_only] [-replace]

Description

Writes out a dofile containing the test points.

Writes out a dofile consisting of add_control_points and/or add_observe_points commands that describe the currently identified test point locations. This dofile can be used to manually edit the list of test points and override the default selections prior to inserting the hardware. It can also be used to completely bypass the analysis step during a subsequent tool invocation.

Arguments

- **-output_file *filename***

  A required switch and string that specifies a name of the file to write the test points information to.

- -all | -control | -observe

  An optional switch that specifies the type of test point to write to the dofile. Choose one from the following:

  -all — Writes out both control and observe points.

  -control — Writes out only the control points.

  -observe — Writes out only the observe points.

- -tool_analyzed_only

  An optional switch that writes out only the tool generated only test points to the dofile. If you do not specify this switch, then the tool writes out both the tool-generated and the manually-added test points.

- replace

  An optional switch that directs the tool to overwrite an existing ***filename***.

Example

The following example performs test point analysis and then writes out the dofile:

```
analyze_test_points
write_test_point_dofile -output my_test_points.dofile -replace
```

Related Commands

add_control_points

add_observe_points

write_visualizer_dofile

Context: unspecified, all contexts

Mode: setup, analysis

Usage

write_visualizer_dofile ***file_pathname*** [-Display *window_name…*] [-Replace]

DFTVisualizer Menu Path:
File > Create Dofile

Description

Writes a dofile containing the commands needed to recreate the current instance and data sets displayed in the DFTVisualizer windows.

The commands in this dofile do not replicate the entire command sequence you have used in the tool session, but rather they simply add the instances and data sets currently present in the specified displays. For example, if you add ten instances to the Design window, delete three of them, then issue the write_visualizer_dofile command, the resultant dofile will contain commands that add seven instances to the Design window.

Arguments

- ***filename***

  A required string that specifies the name of the dofile you are writing.

- -Display *window_name*

  An optional switch and repeatable literal that specify the DFTVisualizer windows to which the dofile should restore current instances and data sets. The valid options for the *window_name* argument, from which you can select as many as you want, are as follows:

  DEBug
  A literal that specifies the Debug window.

  DESign
  A literal that specifies the Design window.

  DAta
  A literal that specifies the Data window.

  Browser
  A literal that specifies the Browser window.

  Signals
  A literal that specifies the Signals window.

  Omit the -Display switch to include commands in the dofile that restore current instance and data displays in the following windows: Data, Debug, and Design.

- -Replace

  An optional switch that specifies to replace the contents of *file_pathname* if a file by that name already exists.

Examples

The following example writes a dofile that will restore instances and data sets currently displayed in the Debug and Design windows of DFTVisualizer:

```
write_visualizer_dofile debug.do -display debug design -replace
```

Related Commands

close_visualizer
open_visualizer
set_visualizer_preferences

write_visualizer_preferences

Context: unspecified, all contexts

Mode: setup, analysis

Usage

write_visualizer_preferences ***filename***

DFTVisualizer Menu Path:
Edit > Preferences: Save Preference File… (button on **Global** tab of DFTVisualizer Preferences dialog box)

Description

Writes the current DFTVisualizer preference settings to the specified file.

To read in the preference settings, use the read_visualizer_preferences command.

Arguments

- ***filename***

  A required string that specifies the name of the text file to which to write the current DFTVisualizer preference settings.

Examples

The following example writes the current DFTVisualizer preference settings to the file, *my_prefs.dftvis*:

```
set_system_mode analysis
write_visualizer_preferences my_prefs.dftvis
```

Related Commands

read_visualizer_preferences

write_window_contents

Context: unspecified, all contexts

Mode: setup, analysis

Usage

write_window_contents ***file_pathname*** [-Replace] **-Display *window_name***
[-Format *file_format*]

DFTVisualizer Menu Path:
File > Save As

Description

Saves the current content of the specified DFTVisualizer window to the specified file.

The window must be visible, not minimized. You can save the file in one of several formats, both graphic and text formats.

To view a graphic file generated by this command, you can use any third-party tool that supports that particular graphic format.

Arguments

- ***file_pathname***

  A required string that specifies the pathname of the screen capture file.

- **-Display *window_name***

  A required switch and literal pair that specifies the DFTVisualizer window whose content you are capturing. Select the window name from the following list:

  DEBug
  A literal that specifies the Debug window. This is the invocation default.

  DESign
  A literal that specifies the Design window.

  Browser
  A literal that specifies the Browser window.

  Signal
  A literal that specifies the Signals window.

  DAta
  A literal that specifies the Data window.

  WAve
  A literal that specifies the Wave window.

  TRanscript
  A literal that specifies the Transcript window.

  Format
  A literal that specifies the Format window.

SEarch
A literal that specifies the Global Search window.

TEst_structures
A literal that specifies the Test Structures window.

- -Replace

  An optional switch that specifies replacement of the contents of *file_pathname*, if a file by that name already exists.

- -Format *file_format*

  An optional switch and literal pair that specify the format of the capture content. By default, JPEG format is used. Format options include:

  schematic
  A literal that specifies the Mentor Graphics Tessent schematic format. This is the default for the Debug and Design windows.

  PS
  A literal that specifies the Postscript format. This format can only be used with the schematic window.

  JPG (or JPEG)
  A literal that specifies the JPG file interchange format. This is the default for all windows except the Debug and Design windows.

  GIF
  A literal that specifies the Graphics Interchange Format.

  PNG
  A literal that specifies the Portable Network Graphics format.

  TXT
  A literal that specifies ASCII text format. This option is valid for the Browser, Data, and Signal windows only.

  CSV
  A literal that specifies the CSV (comma-separated values) file format. This option is valid for the Browser, Data, and Global Search windows.

Examples

The following example analyzes a DRC violation (automatically invoking DFTVisualizer in the process and displaying the part of the netlist related to the violation in the Debug window), then saves a screen capture of the displayed netlist segment in GIF format:

analyze_drc_violation c3-1

```
//  command: open_visualizer -display debug
//  Note: Gate report now set to clock_cone (clock=/tck).
//  Creating Schematic for 5 instances
```

write_window_contents my_c3-1.gif -display debug -format gif

```
//  INFO: "debug window" contents saved in my_c3-1.gif (10599 bytes) using
//  the "gif" format.
```

Related Commands

close_visualizer
open_visualizer
read_window_contents
set_visualizer_preferences

Chapter 8
Design Rule Checking

Design Rule Overview 2135
Design Rules 2147

Design Rule Overview

The following design rules are described in this chapter:

- RAM Rules (A Rules)
- BIST Rules (B Rules)
- Clock Rules (C Rules)
- Scan Cell Data Rules (D Rules)
- Extra Rules (E Rules)
- EDT Finder Rules (F Rules)
- Flattening Rules (FN, FP, and FG Rules)
- General Rules (G Rules)
- ICL Extraction Rules (I Rules)
- ICL Semantic Rules (Semantic Rules)
- EDT Rules (K Rules)
- Procedure Rules (P Rules)
- Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)
- Scannability Rules (S Rules)
- Scan Chain Trace Rules (T Rules)
- Power-Aware Rules (V Rules)
- Timing Rules (W Rules)

Scan Insertion Checking

Prior to scan insertion, the tool performs limited rule checks on the design as you switch from setup to analysis mode. Part of the checking it does is scannability checking. For more information, refer to Scannability Rules (S Rules).

For primary clock inputs gated by other logic, a test procedure file describes the logic conditions that permit propagation of the clock signal through these gates. For uncontrollable clock circuitry, the tool can assist you in modifying your circuit by inserting test logic circuitry at these clock nodes whenever necessary. Refer to "Enabling Test Logic Insertion" in the *Tessent Scan and ATPG User's Manual* for details.

If you specify existing scan circuitry, or if you have a test procedure file that sets up conditions to allow some state elements to be scan candidates, the tool performs more extensive checking. After you add scan circuitry to your design and generate or write a test procedure file, you should go back to setup mode and specify this information. Then you can return to analysis mode and perform extensive rules checking *before* entering
patterns -scan context.

Troubleshooting Rules Violations

This section provides useful information about handling design rules violations. For information on specific rules violations, refer to "Design Rule Output" on page 2146.

Setting the Handling of Rules

Some rules allow you to specify how a violation is handled either error, warning, note, or ignore. For more information, see set_drc_handling.

Turning on ATPG Analysis

The Atpg_analysis option to the set_drc_handling command provides full test generation analysis during rules checking for clock rules C1, C3, C4, C5, C6, some D rules and some E rules (such as E4, E5, E8, E10, E11 and E12). For the C1 rule, this analysis is enabled by default; for other rules, the tool does not perform the analysis unless you direct it to.

For example, assume that you select Atpg_analysis for clock rule C1 and the tool simulates a clock input to be X. The rule violation occurs when the analysis determines that it is, or may be, possible to turn the clock input on when all other defined clocks are off, and constrained pins are at their constrained values.

Note

The ATPG analysis rules checking process may requires additional CPU time and memory.

Debugging DRC Violations in DFTVisualizer

When you issue the analyze_drc_violation command, the tool automatically opens the DFTVisualizer Debug window (if it is not already open) and sets gate reporting to match the type of rule violation that has occurred.

In the Debug window, DFTVisualizer displays *callouts* next to failing gates or clocks associated with DRC violations. Each callout displays a summary of the DRC and potential next steps. For more information about using callouts, see "Debug Window" in the *Tessent Shell User's Manual*.

Table 8-1 lists the DRCs that are supported by callouts in the DFTVisualizer Debug window and their messages and appearance.

Table 8-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
A1	Failing RAM instance	Setting all write controls off failed to force off RAM write line. Trace write control lines backward to debug the issue further.
A2	Failing RAM instance	Connected to scan clock.
A3	Failing RAM instance	Read or write control line is connected to address input of RAM. Trace address input backwards to debug the issue further.
A4	Failing RAM instance	Read or write control line is connected to data input of RAM. Trace data input backward to debug the issue further.
A6	Failing RAM instance	Write/Read line not off during time <time> of <test proc name> procedure. Refer to the test procedure file to debug the problem further.
A7	Failing RAM instance	Setting all read controls off failed to force off RAM read line. Trace read control lines backward to debug the issue further.
C3	Source of violation	Clocked by same clock/clock_path.
	Failing instance	Captures data from sequential element <source_gate> clocked by same clock </clock_path>.
C4	Source of violation	Clocked by same clock/clock_path.
	Failing instance	Captures data from sequential element <source_gate > clocked by same clock < /clock_path>.
C5	Failing instance	Multiple clock inputs of <clock names> are in the same clock cone and the clock inputs may be on at the same time. Trace back on the clock cone to debug the problem further.
C6	Failing instance	Clock <clock_path> affects both data and clock input.
C7	Failing instance	Clock input </clock_path> cannot capture data with a single clock on.
C8	Primary output pin	Primary output <PO name> is connected to clock <clock name>.
C9	Primary output pin	Path from clock <clock name> is gated by scan cell that uses same clock.

Table 8-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
C16	Failing instance	Load value of scan cell is disturbed by clock input <input_num> at the beginning of the capture phase when pin constraints are forced.
D1	Failing instance	Scan cell disturbed during time <time> procname.
D2	Failing copy instance	Copy cell failed data capture design rule check. See Scan Cell Data Rules (D Rules) for conditions that cause this DRC violation.
D3	Failing Master FF	Not successfully observed by master observe procedure. Refer to the test procedure file to debug the problem further.
D4	Failing Master FF	Skew load procedure not successful for MASTER. Refer to the test procedure file to debug the problem further.
D5	Failing FF	Non-scan memory element converted to <gate type>: TIEX, TIE0, TIE1, INIT-0, INIT-1, INIT-X, and TLA.
D6	Non-scan latch instance	Latch not transparent due to <reason for failure>.
D7	Failing instance	Clock port set to stable high at the end of shift procedure.
D8	Failing Master latch	Master latch allows data capture while clocks off. Refer to the test procedure file to debug the problem further.
D9	Failing FF	<Type of disturbance> disturbance in procedure <sequential transparent proc name>. Refer to the test procedure file to debug the problem further.
D10	Failing FF	Invalid transparency (X/Y) in clock procedure <clock proc name>.
D11	Failing FF	Transparent capture cell in procedure <clock proc name> has connectivity to primary outputs.
E7	Failing bidirectional pin	Bidirectional pin <pin name> not set to input mode at time <time> of <test proc name> procedure. Refer to the test procedure file to debug the problem further.
F2	Failing gate	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F5	Failing gate	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
	Failing channel input	Channel input is not driving any decompressor.

Table 8-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
F6	1st Failing state element	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
	2nd Failing state element	Channel input is not driving any decompressor.
F7	Failing state element	Driven by a primary input which is not defined as a channel input in the dofile.
	Channel input	Not defined as a channel input in the dofile, but drives a decompressor. Refer to the dofile or the netlist to debug the problem.
F8	Channel out port	The channel output is not driven by a compactor. Refer to the netlist or trace the schematic to debug the problem further.
	Channel in port	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F9	Failing state element	Trace stopped here. Refer to the netlist or trace the schematic to debug the problem further.
F12	Scan in pin	Scan in pin of the uncompressed scan chain was not defined using “add scan chains” command.
	Scan out pin	Scan out pin of the uncompressed scan chain was not defined using “add scan chains” command.
F13	Failing state element	State element is not reset correctly prior to shift. Refer to the netlist or test procedure file to debug the problem further. Simulated Value = Expected Value =
F14	Failing state element	State element does not hold its value during capture. Refer to the netlist or test procedure file to debug the problem further.
F15	Failing state elements in the path	Initialized to wrong value <val> after test_setup and load_unload. Expected value is <val>. Refer to the netlist or test procedure file to debug the problem further.
F16	Failing state element	State element does not preserve its value during capture and load_unload. Refer to the netlist or test procedure file to debug the problem further.
F17	Failing state element	Failing state elements of the MISR.

Table 8-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
F18	Failing Mask hold register	State element does not load expected values mask shift register before shift in load_unload. Refer to the netlist or test procedure file to debug the problem further. Simulated Value = <val> Expected Value = <val>
	2nd Failing state element	The driving mask shift register.
F19	Failing state elements	Does not preserve its value during shift. Refer to the netlist or test procedure file to debug the problem further.
I1	Failing instance	Ports found in ICL module are not found on corresponding design module.
I2	Failing instance	ICL port does not have a valid direct path to or from another ICL port function.
I3	Failing instance	ICL port is connected to an incompatible ICL port function.
I4	Failing instance	ICL port is connected to a primary input or output port previously inferred to be an incompatible ICL port function.
I5	Failing instance	Input constraint verification failed for an input pin on an ICL-attributed design instance.
P92	Failing gate	Internal primary input <path> is not controlled.
R2	Scan input pin	Tracing back from this scan input pin/scan_input_path could not reach a top-level port.
	Top-level gate	Could not trace further back from this gate.
R4	Failing pin	This pin had an input constraint of expected value (core module_ name mode internal/external) but currently has value (actual Val).
R5	Failing instance	The stable_after_setup value of pin was expected val but currently has an actual value.
V8	Scan cell	Power Domain: < name>, Power Mode (CPF) or Power State: <name>.
V9	Scan cell	Power Domain: < name>, Power Mode (CPF) or Power State: <name>.
V13	1st failing gate	Power Domains <power_domain_name>.
	2nd failing gate	Power Domains <power_domain_name>.

Table 8-1. DRC Violation Messages

DRC Rule	Instance or Port Callout Box is Attached To	Callout Message Displayed
V14	1st failing gate	Power Domains <power_domain_name>.
	2nd failing gate	Power Domains <power_domain_name>.

Setting the Level of Gate Data

The tools can report data at different levels; therefore, you should specify the level of information before you issue the report_gates command. You do this with the set_gate_level command. Setting the gate level to design (the default) reports information at the design cell (library model) level. Figure 8-1 depicts a scannable-equivalent DFF cell library model at the design level.

Figure 8-1. Example of Design Level

If the gate level is set to design, the report_gates command displays the following information:

```
instance_name        cell_type
    input_pin_name   I   (data)    pin_pathname ...
    input_pin_name   I   (data)    pin_pathname ...
                   . . .
    input_pin_name   O   (data)    pin_pathname ...
```

Setting the gate level to primitive reports information at the simulation gate level. Figure 8-2 depicts the *sdff1* library model at the primitive level.

Figure 8-2. Example of Primitive Level

If you set the gate level to primitive, the report_gates command displays the following information:

```
instance_name        (gate_id#)    gate_type
    input_pin_name   I   (data)    gate_id#-pin_pathname ...
    input_pin_name   I   (data)    gate_id#-pin_pathname ...
                     . . .
input_pin_name   0   (data)   gate_id#-pin_pathname...
```

In the DFTVisualizer Debug window, you can change between the Primitive and Design levels by pressing **Ctrl + L**, or by selecting **Display** > **Gate Level (Ctrl + L)** > **Design | Primitive**.

Setting the Gate Information Type

The set_gate_report command specifies the type of information that you want to appear when you report_gates data with the report_gates command. The multitude of options this command supports varies somewhat depending on which tool you are using. The common usage of the set_gate_report command is:

set_gate_report {**Normal** | **Trace** | **Error_pattern** | **TIe_value** | **Constrain_value** | {**Drc_pattern** ***procedure_name*** [*time* | -All]}

- The Trace option displays the values of the gates obtained during scan chain tracing. That is, this option displays data obtained on an error condition (not warning) during simulation of the **shift** procedure. You can use this option to help determine why a scan chain was not properly sensitized.
- The Error_pattern option displays the simulated values of the gate and its inputs, for the pattern (event) that had an error. This option displays such information as cell disturbances during the **load_unload** procedure or bus contention problems.
- The Normal option is the default. It displays only standard connectivity data.
- The Drc_pattern option displays an identified procedure's simulated gate values during the designated time. This option is similar to the Trace option, but is more

versatile because it allows access to the data obtained from simulation of any of the test procedures.

- The Parallel_pattern option displays simulated values for a selected pattern in the last simulation pass. A “pattern” is any time event that occurs during the test procedure. When the ATPG tool encounters problems in generating patterns, you can access the simulation data with this option.

For information on all the available options or tool-specific uses of this command, refer to the set_gate_report reference page.

Reporting Gate Data

If you encounter rules violations when you attempt to exit the Setup mode, you may need more information about specific gates in the design for troubleshooting purposes. While the violation message may give some information as to the location of the problem, you may need to track down the source of the problem by reporting on a sequence of gates in the design. The report_gates command is a very powerful command you can use to report on netlist data.

The following subsections show how to use report_gates to display various types of information for troubleshooting purposes. For more information on this command, refer to the report_gates reference page.

In DFTVisualizer, you can report data on selected gates by clicking the Report Gates icon, pressing **Ctrl + R**, or pressing the right mouse button and selecting **Commands > Report Gates**. For more information, see “Reporting Gates” in the *Tessent Shell User’s Manual*.

Reporting on a Specific Gate

You can use the report_gates command to display information for selected gates, which you identify by either a gate index number or a pin pathname of a pin connected to the gate. This command reports the gate name, its gate type, and its connectivity to other gates. For example, to use report_gates in this manner, you could specify:

```
report_gates 74493
```

Figure 8-3 shows a report with primitive-level information for a gate with an ID number of 74493.

Figure 8-3. Data Reported for a Specific Gate

Reporting on All Gates of a Specified Type

You can use report_gates to report on all gates of a specified type. The report_gates usage for this case is:

report_gates {-Type *gate_type*}...

The supported gate types are those listed as simulation primitives in the "Simulation Primitives of the Flattened Model" section of the *Tessent Scan and ATPG User's Manual*.

The following example shows how to report on all TIE0 gates.

report_gates -type tie0

```
//  ---------------------------------------------------------
//  List of TIE0 gates
//  ---------------------------------------------------------
//  /u1/inst__565_ff_d_0__dff (13)  TIE0
//      "OUT"  O  267-  266-
//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
//  /u1/inst__565_ff_d_2__13 (15)  TIE0
//      "OUT"  O  271-  270-
//  Total number of tie0 gates = 3
```

Reporting a Histogram of All Gate Types

You can use report_gates to show a distribution (histogram) of all gates in the design. To use report_gates in this manner, specify:

report_gates -type Histogram

The following example shows the type of data this command displays.

```
//  ---------------------------------------------------------
//  List of histogram of gates
//  ---------------------------------------------------------
//  BUF=175  INV=30  AND=3  NAND=17  OR=7  NOR=5  XOR=2  LA=14
//  PI=12  PO=7  TIE0=7  MUX=7
```

Reporting on a Path Between Two Gates

You can also use report_gates to display information on the circuitry between two specified gates. To use report_gates in this manner, specify:

report_gates -path <gate1_ID#> <gate2_ID#>

Reporting on the First Input of a Gate

The report_gates command can display data on the gate connected to the first input of the previously reported gate. This lets you quickly and easily trace backward through circuitry. To use report_gates in this manner, first report on a specific gate and then enter:

back

The following example shows how to use report_gates and back to trace backward through the first input of the previously reported gate.

report_gates 26

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

back

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

back

```
//  /u1/inst__565_ff_d_1__13 (14)  TIE0
//      "OUT"  O  269-  268-
```

Reporting on the First Fanout of a Gate

Similar to tracing backward through circuitry, you can also trace forward through the first fanout of the previously reported gate. To use report_gates in this manner, first report on a specific gate and then enter:

forward

The following example shows how to use report_gates and forward to trace forward through the first fanout of the previously reported gate.

report_gates 269

```
//  /u1/inst__565_ff_d_1__13 (269)  LA
//      "S"    I  14-
//      "R"    I  145-
//      SCLK   I  4-/clk
//      D      I  265-/u1/_g32/X
//      ACLK   I  2-/scan_mclk
//      SDI    I  20-/u1/inst__565_ff_d_0__dff/Q2
//      "OUT"  O  26-  27-
```

forward

```
//  /u1/inst__565_ff_d_1__13 (26)  BUF
//      "I0"   I  269-
//      "OUT"  O  268-  75-
```

forward

```
//  /u1/inst__565_ff_d_1__13 (268)  LA
//      "S"    I  14-
//      "R"    I  145-
//      BCLK   I  1-/scan_sclk
//      "D0"   I  26-
//      "OUT"  O  24-  25-
```

Related Commands

- report_drc_rules— Displays data associated with violated rules.
- set_trace_report— Displays all scan chain gates traced during rules checking.

Flattening Rule Violations

If you encounter flattening rule violations, you can use the report_flattener_rules command to display either a summary of the flattening rule violations or the data for a specific violation, for troubleshooting purposes. For more information on this command, refer to the report_flattener_rules command.

You can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate rules. For more information on this command, refer to the set_flattener_rule_handling command.

Design Rule Output

Many error messages provide a line number and file name to help you resolve the error condition. However, some error messages may relate to problems with internally generated data and do not provide a line number and file name.

For example, a syntax error (P1) in a test procedure file would produce the following message:

```
The following occurred at line 10 in file testproc
Syntax error in line number 10. (P1-1)
```

Design Rules

This section lists the DRC classes and each of the DRCs in that class.

RAM Rules (A Rules)

The tool checks RAM gates to identify proper test methods. You can select the handling of any RAM rule to be error, warning, note, or ignore. The following subsections describe each of the RAM rules.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

A1

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

When all write control lines are at their off-state, all write, set, and reset inputs of RAMs must be at their inactive state. The tool performs this check using the simulated values that result when all defined write control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any write, set, or reset input of any RAM gate is not off.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed on the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write control lines (including those that are RAM set and reset lines) or defining the wrong off-state.

The occurrence message is:

```
Write controls off failed to force off RAM T line of N (G).  (A1-1)
```

T is the type of input (write, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, and A1-1 is the rule and violation ID number.

The summary message is:

```
N RAM write/set/reset lines not forced off when write  controls are off.
(A1)
```

N is the number of occurrences of rules violation A1.

A2

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

A defined scan clock must not propagate to a RAM gate, except for its read lines. The tool performs this check by determining the forward cone of influence for all clock pins (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when a RAM gate is in a clock cone.

The default setting for this DRC is “Ignore.” You can enable this check with the set_drc_handling command. When the DRC is enabled and an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Scan clock is connected to RAM N (G). (A2-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAMs are connected to a scan clock. (A2)
```

N is the number of occurrences of rules violation A2.

A3

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A write or read control line must not propagate to an address line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an address line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
RAM control line connected to address input of RAM N (G).  (A3-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM address inputs are connected to a RAM control line.  (A3)
```

N is the number of occurrences of rules violation A3.

A4

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A write or read control line must not propagate to a data line of a RAM gate. The tool performs this check by determining the forward cone of influence for all write and read control lines (write/read cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained value, and the initialized non-scan cells are set to their stable stated. The rule violation occurs when a data-in line of a RAM gate is in the write/read cone.

The default handling for this rule violation is warning. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, W indicates write/read control line cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
RAM control line connected to data input of RAM N (G). (A4-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM data inputs are connected to a RAM control line. (A4)
```

N is the number of occurrences of rules violation A4.

A5

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A RAM gate must not propagate to another RAM gate. The tool performs this check by determining the forward cone of influence for all RAM gates (RAM cone). The bounds for the cone of influence are scan cells, RAM gates, and circuitry set to a fixed value when constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The rule violation occurs when an input of a RAM gate is in the RAM cone.

The default handling for this rule violation is warning. The occurrence message is:

```
RAM N1 (G1) connected to RAM N2 (G2). (A5-1)
```

N1 is the instance name of one RAM gate, G1 is its gate ID number, N2 is the instance name of the other RAM gate, and G2 is its gate ID number.

The summary message is:

```
N RAMs are connected to RAMs. (A5)
```

N is the number of occurrences of rules violation A5.

A6

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

All the write inputs of all RAMs and all the read inputs of all data_hold RAMs must be at their off-state during all test procedures, except **test_setup**. The tool performs this check using the simulated values that result when it simulates the test procedures. The rule violation occurs if any write, set, or reset input of any RAM, or a read input of a data_hold RAM is not off.

The default handling for this rules violation is warning. Failure to satisfy this rule for write inputs results in the RAM being unavailable to hold its contents during scan operation, which may cause a loss in test coverage. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all write or read control lines, or defining the wrong off-state.

The occurrence message is:

```
L line of RAM N (G) not off during time T of P procedure.  (A6-1)
```

L is the type of input (write, read, set, or reset), N is the instance name of the RAM gate, G is the gate ID number, T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of uncontrolled RAMs during test  procedures.
(A6)
```

N is the number of occurrences of rules violation A6.

A7

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

When all read control lines are at their off-state, all read inputs of RAMs with the read_off attribute set to hold must be at their inactive state. The tool performs this check using the simulated values that result when all defined read control lines are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs if any read input of a data_hold RAM gate is not off.

The default handling for this rules violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the input that is not held off, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining all read control lines or defining the wrong off-state.

The occurrence message is:

```
Read controls off failed to force off RAM read line of N (G).  (A7-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N RAM read lines not forced off when read controls are off.  (A7)
```

N is the number of occurrences of rules violation A7.

A9

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A read-only RAM model should define an initialization file. If an initialization file is not defined, the tool will treat instances of the RAM as TIEX, reducing test coverage.

The default handling for this rule violation is warning. To correct the problem, be sure the ATPG library model of the RAM includes an **init_file** attribute that correctly specifies an initialization file.

The occurrence message is:

```
The initialization file is not defined for read-only RAM N (G).(A9-1)
```

N is the instance name of the RAM gate and G is the gate ID number.

The summary message is:

```
N read-only RAMs do not have initialization file. (A9)
```

N is the number of occurrences of rules violation A9.

A10

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The content of a read-only RAM that does not have a write port must not be disturbed by any procedures except test_setup.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during simulation of a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not forcing the primary inputs, or not initializing the state elements, that control the RAM's set or reset port to an appropriate value.

When the A10 and/or A11 rule is violated, the tool will treat the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

```
The content of read-only RAM N (G) can be disturbed through L line during
time T of P procedure. (A10-1)
```

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (set or reset), T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of disturbed read-only RAMs during test
procedures.  (A10)
```

N is the number of occurrences of rules violation A10.

A11

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The content of a read-only RAM that does not have a write port must not be disturbed through the set or reset lines during capture.

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not constraining the primary inputs or the state elements that control the RAM's set or reset port.

When the A10 and/or A11 rule is violated, the tool will treat the RAM as TIEX, which may reduce fault coverage.

The occurrence message is:

```
The content of read-only RAM N (G) can be disturbed through L line in
capture window.(A11-1)
```

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (set or reset).

The summary message is:

```
There were N occurrences of disturbed read-only RAMs in capture window.
(A11)
```

N is the number of occurrences of rules violation A11.

A12

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The content of a read-only RAM that has a write port must not be disturbed by any procedure except test_setup (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during a procedure, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

When rule A12 and/or A13 is violated, the tool will ignore the RAM's contents initialized through the initialization file.

The occurrence message is:

```
The initialization file of RAM N (G) cannot be used during test pattern
generation since RAM content can be disturbed through L line during time T
of P procedure.(A12-1)
```

N is the instance name of the RAM gate, G is the gate ID number, L is the type of input (write, set or reset), T is the time, and P is the procedure name.

The summary message is:

```
There were N occurrences of disturbed RAM initialization file during test
procedures.  (A12)
```

N is the number of occurrences of rules violation A12.

A13

Category: RAM

Contexts Supported: dft -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The content of a read-only RAM that has a write port must not be disturbed through the write, set or reset lines during capture (in order for the initialization file to be usable for test generation).

The default handling for this rule violation is warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the set or reset port that is not held off during capture, and by tracing back from this input, you can identify how to correct the problem. The usual cause of this error condition is not defining the write, set or reset operation properly.

When rule A12 and/or A13 is violated, the tool will ignore the RAM's contents initialized through the initialization file.

The occurrence message is:

```
The initialization file of RAM N (G) cannot be used during test pattern
generation since RAM content can be disturbed through L line in capture
window. (A13-1)
```

N is the instance name of the RAM gate, G is the gate ID number, and L is the type of input (write, set or reset).

The summary message is:

```
There were N occurrences of disturbed RAM initialization file in capture
window. (A13)
```

N is the number of occurrences of rules violation A13.

A14

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be observable. The tool performs this check by forcing all pin constraints and tied cells and check if there is any sensitizable path from the outputs of a RAM to any observation point. The rule violation occurs if there is none.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) is unobservable. (A14-1)
```

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints blocking the output of the RAM.

A15

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to read. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the read enable and read clock lines of each read control port of a RAM. The rule violation occurs if the simulation values indicate that all read control ports are off.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot read because its read control ports are
// constrained off. (A15-1).
```

N is the instance name of the RAM gate and G is its gate ID number.

This identifies the RAM gate that cannot be observed. The usual cause of this error condition is constraints that prevent the read enable or read clock lines of a RAM to be turned ON.

A16

Category: RAM

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

RAM should be able to write. The tool performs this check by forcing all pin constraints and cell constraints and check the simulation values at the write enable and write clock lines of each write control port of a RAM. The rule violation occurs if the simulation values indicate that all write control ports are off. This rule checking will not be performed for read-only ROMs.

The default handling for this rule violation is a warning. When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and use the report_gates command for the gate ID number displayed on the error message.

```
// Warning: RAM N (G) cannot write because its write control ports are
// constrained off. (A16-1).
```

N is the instance name of the RAM gate and G is its gate ID number.

BIST Rules (B Rules)

Whenever LFSRs are defined, the tool performs BIST rules checking to ensure proper application of BIST patterns to the circuit. You cannot change the handling of the BIST rules from their default conditions of either error or warning—with the exception of rule B2. The following subsections describe each of the BIST rules.

B2

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

Every scan chain input pin must connect to an LFSR. Correct this error condition by connecting the scan chain input pin of the indicated chain to an LFSR. You can change how the rules checker handles this check with the set_drc_handling command.

The error message is:

```
Input of chain C has no LFSR connection. (B2-1)
```

C is the name of the scan chain.

B4

Category: BIST

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

Every scan chain output pin must connect to an LFSR. Correct this error condition by connecting the scan chain output pin of the indicated chain to an LFSR.

The error message is:

```
Output of chain C connected to parallel shift LFSR. (B4-1)
```

C is the name of the scan chain.

Clock Rules (C Rules)

The application checks the scan clocks to ensure their proper definition and operation. You may select the handling of any clock rule to be error, warning, note, or ignore. The following subsections describe the clock rules and the special handling you can set for them.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

Clock Terminology

The clock rule information contains a couple of recurring concepts. To make optimal use of the information, you should understand these concepts as described in the following subsections.

Clock Signals

The tool considers any signal to be a clock if it can change the state of a sequential element, including system clocks, sets, and resets. When you define each clock primary input (a required setup step you perform using the add_clocks command or "analyze_control_signals -Auto_fix"), a key piece of information specified with either of these commands is the pin value that represents the clock's off state. Two important terms arise out of this definition, as shown in Figure 8-4.

Figure 8-4. Clock Cycle Terminology

The transition of the clock from the off state to the on state is considered the leading edge of the clock, while the transition from the on state to the off state is considered the trailing edge of the clock.

Clock Cones and Effect Cones

A gate pin or output pin is considered to be in the *clock cone* of a clock signal when they are connected through combinational logic gates and transparent latches (TLAs) to the clock primary input.

Note

The tools treat a latch as a transparent latch (TLA) rather than as a normal level-sensitive sequential element (LA) if it will be continuously enabled at the start of the capture cycle when all clocks are at their defined off values. This treatment therefore depends on tool setups and how the latch is wired into the design. A TLA passes values without holding state and thus acts like a buffer. For more information, refer to "Simulation Primitives of the Flattened Model" in the *Tessent Scan and ATPG User's Manual*.

The clock cone is basically the fanout of the clock signal through strictly combinational logic and TLAs. Pins B, C, CLK and CLK2_INV in the circuit excerpt shown in Figure 8-5 are all in CLK2's clock cone.

Figure 8-5. Example Clock Cone

The tools consider a gate pin or output pin to be in a clock's *effect cone* if there is a sequential element between the clock net and the gate pin or output pin. In Figure 8-6, pin Y is in the effect cone of CLK2.

Figure 8-6. Example Effect Cone

A pin can be in both the *clock cone* and the *effect cone* as shown for pin D on one of the flip-flops in Figure 8-7.

Figure 8-7. Pin in Both Cones

You can get the Debug window of DFTVisualizer to show a C, E or B near device pins that are in the clock (C), effect (E), or both (B) cones of a particular clock by issuing the set_gate_report command with the Clock_cone option after the tool has flattened the design. For example:

```
create_flat_model
set_gate_report clock_cone CLK
```

Alternatively, you can choose **Data** > **Clock Cone** from the DFTVisualizer menu (with the Debug window active) and in the dialog box specify the clock. Figure 8-8 shows an example of how this might look for a circuit fragment in the Debug window of DFTVisualizer.

Figure 8-8. DFTVisualizer Debug Window View Showing Clock Cones

When the Clock_cone option is in effect, the output of the report_gates command also displays clock cone data:

```
report_gates /DFF1

//  /DFF1  dff
```

```
//    clkI  (C)2-/CLK
//    d  I  (-)1-/C
//    q  O  (E)5-
```

Note

The tool shows clock and effect cone data for just one clock at a time—the clock whose primary input *pin_name* you specified in the last “set_gate_report Clock_cone” command.

The ATPG Analysis Option

Clock rules C1, C3, C4, and C5 can run full ATPG analysis during their checks. For more information on ATPG analysis, refer to “Turning on ATPG Analysis” on page 2136.

C1

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A scan or non-scan cell must not capture data when all specified clocks are set to their off states.

When all clocks are at their off state as defined with the add_clocks command, all clock inputs (including sets and resets) of scan and non-scan cells must be at their off state. For non-scan cell violations, the tool converts these to TIEX.

In the case of a DFF scan cell, the clock input must be a stable binary value (0 or 1).

Note

If the clock input value of a DFF scan cell simulates as X, the tool by default performs additional analysis to determine if the X represents a stable binary value. An X that represents a stable binary value satisfies the C1 rule's clock input requirement for a DFF. (To turn off the additional analysis, use the set_drc_handling command's Noatpg_analysis argument.)

The tool performs this check using the simulated values that result when all defined clocks are at their defined off states, constrained pins are set to their constrained values, initialized non-scan cells (initialized during test_setup or load_unload) are set to their stable states, and C0, C0DX, C1, and C1DX cell constraint values are applied. The rule violation occurs if any clock input, including set and reset lines, of any scan or non-scan cell memory element is not at its off state.

When the C1 violation occurs on a set/reset port or a level-sensitive latch port, the tool does not issue an error message and does not stop the DRC process. Instead, the tool issues the following summary messages to indicate such situation exist:

```
// Note: n scan cells whose clock ports are not off when all clocks are
// off will be handled through additional ATPG effort.

// The scan cells have been excluded from the list of C1 DRC violations.
// Use "report_drc_rules C1 -excluded" to report these C1 violations
```

If the number of such scan cells is large, ATPG runtime is increased. One common root cause of these C1 violations is a set/reset pin not defined as a clock. You can define a set/reset pin as a clock to fix the problem and avoid ATPG performance impact.

Figure 8-9 shows an example of a circuit segment that produces a C1 violation due to an indeterminate value (X) at the clk input of DFF1 when the off state of the clocks at the PIs is defined as a logic 0 and all defined clocks are off.

Figure 8-9. C1 Violation Example

The clock input value to the scan cell is X because the EN signal value is X. If EN is left at X, the signal arriving at the clk input of the scan cell cannot be held off.

Effect on Testability

Failure to satisfy this rule can result in unstable scan and non-scan cell values going into or coming out of load_unload, or between cycles, and can result in inaccurate test pattern simulation results (simulation mismatches) during verification in a timing-based simulator.

How to Debug C1 Violations

The occurrence message is:

```
// Clock PIs off failed to force off a clock line of T N (G). (C1-1).
```

T is the type of scan cell (MASTER, SLAVE, etc.), N is the instance name of the gate, and G is the gate ID number.

The summary message is:

```
// There were N clock rule C1 fails (unstable scan cells when clocks off).
```

N is the number of occurrences of rules violation C1.

For debugging, be sure C1 handling is set to error (the invocation default setting) before you enter a “set_system_mode analysis” command to leave setup mode. Use the following command to check the setting:

report_drc_rules -fails_summary

If the handling is other than error, you can change it back to error with:

set_drc_handling c1 error

When C1 handling is set to error, the first C1 violation that occurs will stop the DRC process and return the tool to Setup mode.

Tip: The C1 violation on which the tool stops is the C1 you must debug. Fixing the C1 violations the tool singles out in this manner will often fix other C1’s with no extra effort on your part, so is very efficient.

The tool will transcript details about the violation and save the simulation data related to the violation, so you can report it.

You can then debug the violation using DFTVisualizer or by issuing commands from the tool’s command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C1 DRC error using DFTVisualizer, use the following command:

open_visualizer

Then choose **Tools** > **Analyze DRC** from the DFTVisualizer menu, select the C1-1 ID of the C1 error occurrence in the dialog box and click Analyze. The scan cell affected by that occurrence is displayed in the Debug window.

Tip: Alternatively, you can issue the analyze_drc_violation command with the C1-1 argument at the tool command line. For example:

analyze_drc_violation c1-1

Once the offending scan cell is displayed, trace backward from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the add_clocks command, is correct. To change the defined off state, delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

How to Debug from the Tool Command Line

To view the location of a C1 DRC error from the tool command line, use the following example command steps:

1. set_drc_handling C1 Error
2. set_gate_report Error_pattern
3. set_system_mode analysis
4. report_drc_rules C1-
5. report_gates *offending_gate's_id#*
6. *report_gates* -Endpoints -Backward *id#_of_gate_connected_to_offending_gate's_clock*

The following transcript excerpt shows an example of the use of this command sequence starting at step 4:

```
 report_drc_rules c1-1
// Error: Clock PIs off failed to force off a clock line of /U$1/DFF1
// (14). (C1-1)
 report_gates 14
//  /U$1/DFF1 (14) DFF
//       "S"   I  (0) 10-
//       "R"   I  (0) 11-
//       clk   I  (X) 12-/OR/Y
//       d     I  (X) 13-/MUX/out
//       "OUT" O  (X) 8-   9-
 report_gates -endpoints -backward 12
//  ---------------------------------------------------------------------
//  Begin backward trace for gate /OR (12).
//  ---------------------------------------------------------------------
//  /CLK (3)  PI
//      CLK     O  (0)  12-/OR/A0
//  /EN (4)  PI
//      EN      O  (X)  12-/OR/A1
//  Number gates in trace = 2.
```

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. If the off state is correct, the problem typically is due to an indeterminate value (X) in related logic that causes the clock's value at a scan cell to be X. This is the case in the example shown in the figure above. You can fix these by using the add_input_constraints command to constrain a primary input pin to a value that removes the X:

```
add_input_constraints -c0 EN
```

This allows the tool to consider the CLK signal to be the only clock signal to the clk input of the scan cell.

For further explanation and examples of possible C1 rule violations, refer to the "Clock Gaters" appendix in the *Tessent Scan and ATPG User's Manual*.

Related Commands

add_clocks
add_input_constraints
analyze_drc_violation
delete_clocks
report_clocks
report_drc_rules
report_gates
set_gate_report

C2

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Error for patterns -scan context. Warning for dft -scan and dft -test_points contexts.

report_drc_rules: Supported

Description

Each clock must have a structural path to the clock port of at least one memory element. The application performs this check by a backward tracing from the clock port, SET/RESET port, and read/write enables of all memory elements toward primary inputs. The rule violation occurs if a clock primary input cannot be reached.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. Defining a pin to be a clock, when it does not behave as a clock, is the most usual cause of this error condition. The default handling for this rule violation in the patterns -scan context is error. You can tell the system to ignore this error condition by using the -Force switch of the set_system_mode command. Failure to satisfy this rule indicates a defined clock cannot capture data, thus reducing test coverage.

When a floating pin is defined as a clock, it is suppressed from C2 violation when the design also defines some internal clocks. This exception prevents false C2 violations for a design with PLL clocking where the external reference clock is disconnected from the netlist in the flat model.

The occurrence message is:

```
Clock P cannot reach the clock port of any memory element. (C2-1)
```

P is the pin name of the clock.

The summary message is:

```
There were N clock rule C2 fails (reachability check).
```

N is the number of occurrences of rules violation C2.

C2 Rule Violation Example

Figure 8-10 shows an example circuit and circuit setup specified in the tool.

Figure 8-10. C2 Rule Example Circuit

If you run rules checking on this design, you will get a C2 rules violation because while the CK17 signal appears to be a clock (due to its name), it cannot be reached from the clock port or SET/RESET ports of the flip-flop. To fix this problem, add the command:

delete_clocks CK17

Then, you must re-run checks.

C3

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Description

When a sequential element (or RAM) source and sink are clocked by the same clock and the sink captures data from the source, a potential exists for the captured data to pass through both the source and sink in the same clock cycle. That is, the sink might capture the source's new data (captured in the same cycle) instead of the source's old data (captured in the previous cycle). Table 8-2 lists four clocking relationships that can produce this problem.

Table 8-2. Clocking that Can Result in a C3 Signal Race

Source	Sink
LS	LS
LS	TE
LE	LS
LE	TE

This condition violates the tool's default assumption that the sink captures only old data from the source, so results in a C3 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Note

Adequate special handling for most C3 situations is to simply issue "set_split_capture_cycle On" as explained under "Possible Resolutions" on page 2181.

Figure 8-11 shows an example DFTVisualizer Debug window display of a circuit segment that produces a C3 violation when the off state of the clock at the PI is defined as a logic 0. (In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups). DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock's trailing edge (TE) due to the inverter. Under certain timing conditions (if captured data propagates through DFF1 to its q output in less than half a clock cycle for example), data will pass through both DFF1 and DFF2 in a single clock cycle.

Note

The same situation would occur if the clock pin was connected directly (rather than through an inverter) to DFF2, and DFF2 was negative-edge triggered.

Figure 8-11. C3 Violation Example

Figure 8-12 shows a possible timing scenario for the preceding circuit segment.

Figure 8-12. Example Timing That Allows Sink to Capture Source's New Data

The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones). For an introduction to clock and effect cones, refer to "Clock Cones and Effect Cones." The bounds for the clock cones are scan cells and circuitry set to a fixed value when the

constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states. The clock cone stops at read ports of RAMs that have the read_off attribute set to hold, and then the effect cone propagates from its outputs.

Note

In Figure 8-11, the data input for DFF2 is in the effect cone of CLK and its clock input is in the clock cone of CLK as indicated by the E and the C in the Debug window display of DFTVisualizer.

The rule violation may occur for a clock if one of the following is true:

- The clock input of a DFF state element is in the clock cone, its input data is in the effect cone, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in the clock cone and its input data is in the effect cone.
- The write input of a RAM is in the clock cone and a data input or address input of the associated write port is in the effect cone.
- The read input of a RAM is in the clock cone and an address input of the associated read port is in the effect cone.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the set_drc_handling command with the Atpg_analysis argument. By default, the rules checker performs a partial ATPG analysis to find potential C3 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to “Screening Out False C3 Violations” on page 2182.

Effect on Testability

Failure to satisfy this rule can result in inaccurate test pattern simulation results (simulation mismatches) during verification in a timing-based simulator and failing patterns on the tester.

How to Debug C3 Violations

For C3 violations, the tool transcripts a message similar to the following:

```
// Note: There were N clock rule C3 fails (clock may capture data
            affected by its captured data).
```

N is the number of times a C3 violation occurred.

Use the report_drc_rules command to obtain additional information about the violations. For example, to report the occurrence messages for all the C3s, issue “report_drc_rules C3.” The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with “C3-” and the occurrence number. For example:

report_drc_rules c3-1

```
// Note:    Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
//          Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C3 violation using DFTVisualizer or by issuing commands from the tool’s command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C3 DRC violation using DFTVisualizer, use the following command steps:

1. set_system_mode analysis
2. Choose **Tools** > **Analyze DRC** from the DFTVisualizer menu and select the desired C3 DRC violation in the dialog box. The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Debug window as shown in Figure 8-11.

Tip: Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool’s command line. For example:

analyze_drc_violation c3-1

Note

In some situations, the tool’s analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the -Interval switch with the set_drc_handling command.

How to Debug from the Tessent Tool Command Line

To view the location of the first occurrence of a C3 DRC violation from the tool command line, use the following example command steps:

1. set_system_mode analysis
2. report_drc_rules C3-*occurrence#*
3. set_gate_report Clock_cone *pin_name*
4. report_gates *source_gate_id sink_gate_id*

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

```
report_drc_rules c3-1
// Note: Clock /CLK failed rule C3 on input 3 of /DFF2 (9). (C3-1)
// Source of violation: input 3 of /DFF1 (8)
set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2
//  /DFF1  dff
//    "IO"I (-)4-
//    "I1"I (-)3-
//    clkI  (C)2-/CLK
//    d  I  (-)1-/C
//    "OUT"O(E)5-
//  /DFF2 (9) dff
//    "IO"I (-)4-
//    "I1"I (-)3-
//    clkI  (C)7-/INV/out
//      dI  (E)5-/DFF1/q
//    "OUT"O(E)6-
```

Possible Resolutions

Note

There are special cases where C3 rule violations do not cause problems and can be ignored. Although these cases are rare, be sure you understand how your circuit behaves versus what the tool simulates before deciding to ignore C3s.

There are two methods you can use to assure that valid C3 rule violations do not result in simulation mismatches:

- Split Capture Cycle

  The first and preferred method is to enable the set_split_capture_cycle command (the tool enables it automatically if you use create_patterns). This forces a second evaluation of the cells affected by C3 DRC violations. The additional evaluation allows the new data to propagate and in turn be captured into the sink gate. If additional evaluation is required, a side effect of this command is an increase in simulation run time. If the

increase in run time is excessive, you may want to use the other method, capture handling.

Note

When you use “set_split_capture_cycle on,” you will still get C3 rule violations but you can ignore them.

- Capture Handling

  The second method of addressing C3 DRC violations is to use the set_capture_handling command. A side effect of this command is reduced test coverage if the number of C3s is significantly high. The item to consider is that only one simulation value is used for a given source. As a result, the sink gates can only capture either new or old data. Given the case where a source drives two sinks and one sink captures old data and the other sink captures new data, one of the sinks must capture X. The advantage of the set_capture_handling command is there is no impact on ATPG run time.

Mentor Graphics recommends you use “set_split_capture_cycle on.” In cases where the impact on run time is severe and the number of C3 violations is small, then the use of capture handling is recommended. Do not use both methods simultaneously, however.

Screening Out False C3 Violations

For performance reasons, the tool’s default DRC may occasionally report a false C3 violation. There are three additional analyses you can do to screen out these false C3s:

- Use the set_drc_handling command with the Atpg_analysis option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the -Mode A option to the set_drc_handling command.

  When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

  - **Inactive low (IL)**
    When the selected clock is low, the clock port of the latch is inactive.
  - **Inactive high (IH)**
    When the selected clock is high, the clock port of the latch is inactive.
  - **Active high slave (AHS)**
    When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input

of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

- **Active low slave (ALS)**
  When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting with "set_learn_report On," you can then use report_gates to report on the individual latches in these categories.

You can screen out false violations of the C3 rule by issuing the set_drc_handling command before rules checking. The command usage in this context is:

set_drc_handling C3 [-Mode A *clock_name*]

The tool ignores violations of the C3 rule if the following conditions are true:

- The failing latch port is IL or AHS, and the source latch port is IH or ALS.
- The failing latch port is IH or ALS, and the source latch port is IL or AHS.
- All clock, set, and reset inputs of the failing latch are low in the case that all defined clocks are off, the violation source clock is high, and all other clock, set, and reset inputs of the source latch are low.

Related Commands

add_capture_handling
report_gates
set_capture_handling
set_drc_handling
set_gate_report
set_learn_report
set_split_capture_cycle

C4

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

When a sequential element (or RAM) source and sink are clocked by the same clock and the output of the source is connected through combinational logic to the clock port of the sink, a potential exists for the source's captured data to alter the clocking of the sink in the same clock cycle. That is, the source's new value (captured in the current cycle) might propagate through the connected logic and affect the clocking of the sink. By default, the tool expects the source's old value (captured in the previous cycle) in the downstream combinational logic.

Table 8-3 lists four clocking relationships that can produce this problem.

Table 8-3. Clocking that Can Result in a C4 Signal Race

Source	Sink
LS	LS
LS	TE
LE	LS
LE	TE

This condition violates the tool's default assumption that the sink's clocking is affected by only old data from the source, so results in a C4 violation and may require special handling. Failure to address this issue can result in simulation mismatches when you verify the test patterns in a timing-based simulator. Each sequential element is either level sensitive (LS), leading edge triggered (LE), or trailing edge triggered (TE).

Note

Adequate special handling for most C4 situations is to simply issue "set_split_capture_cycle on" as explained under "Possible Resolutions" on page 2188.

Figure 8-13 shows an example DFTVisualizer Debug window display of a circuit segment that produces a C4 violation when the off state of the clock at the PI is defined as a logic 0. (In Tessent FastScan and Tessent TestKompress, you define the off state of each clock with the add_clocks command or with analyze_control_signals -Auto_fix as a part of required tool setups).

Figure 8-13. C4 Violation Example

Figure 8-14 shows a possible timing scenario for the preceding circuit segment, where DFF1 updates on the leading edge (LE) of the clock, while DFF2 updates on the clock's trailing edge (TE) due to the gating of the CLK signal with the output of DFF1. Notice that the old value of DFF1/q disables the clock input to DFF2. The tool's default simulation will therefore predict there will be no clock pulse to DFF2 and that it will hold its state as a result. But the leading edge of CLK almost immediately enables the clock gate and DFF2's clock port does receive a pulse. So rather than holding state as the tool expects, DFF2 captures it's D input value on the TE of the now enabled clock gate.

Figure 8-14. Example Where Actual Behavior Differs from Tool's Prediction

The tool performs this check by determining the forward cones of influence for a clock pin (its clock cones) and the forward cones of influence for each scan cell influenced by the clock pin (effect cones). For an introduction to clock and effect cones, refer to "Clock Cones and Effect Cones." The bounds for the cones are scan cells and circuitry set to a fixed value when the constrained pins are set to their constrained values and the initialized non-scan cells are set to their stable states.

Note

In Figure 8-13, the clock input of DFF2 is in both the clock and effect cones of CLK as indicated by the B in the Debug window display. This is the main difference from a C3 violation in which the clock input of the sink is in the clock cone but *not* in the effect cone and the data input is in the effect cone.

The rule violation occurs for a clock if one of the following is true:

- The clock input of a DFF state element is in both the clock and effect cones, the element is trailing edge triggered, and the effect data comes from an element that is leading edge triggered.
- The clock input of a LA state element is in both the clock and effect cones.
- The write input of a RAM is in both the clock and effect cones.
- The read input of a RAM is in both the clock and effect cones.

The tool performs a mutual exclusivity check to determine if the clock/write/read inputs associated with the failure can be active at the same time. To obtain the most benefit from this check, turn on ATPG analysis prior to DRC by issuing the set_drc_handling command with the Atpg_analysis argument. By default, the rules checker performs a partial ATPG analysis to find potential C4 rule violations. This partial analysis justifies clock/data conflicts in the affected circuitry, but stops at decision nodes, RAM, ROM, TIEX, TLA, and all other non-scan state element gates. With complete ATPG analysis explicitly turned on, the rules checker justifies the conflicting values back to PIs or scan cells.

Note

In some situations, violations of this rule may occur when there is no real problem with the design. For information on performing enhanced checking to screen out these false violations, refer to “Screening Out False C4 Violations” on page 2189.

Effect on Testability

Failure to satisfy this rule can result in inaccurate test pattern simulation results (simulation mismatches) during verification in a timing-based simulator and failing patterns on the tester.

How to Debug C4 Violations

For C4 violations, the tool transcripts a message similar to the following:

```
//  Warning: There were N clock rule C4 fails (clock may be affected by its
captured data).
```

N is the number of times a C4 violation occurred.

Use the report_drc_rules command to obtain additional information about the violations. For example, to report the occurrence messages for all the C4s, use:

report_drc_rules c4

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with "c4-" and the occurrence number. For example:

report_drc_rules c4-1

```
//Warning:  Clock /CLK failed rule C4 on input 3 of /DFF2 (9). (C4-1)
//          Source of violation: input 3 of /DFF1 (8).
```

You can debug a specific occurrence of a C4 violation using DFTVisualizer or by issuing commands from the tool's command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a C4 DRC violation using DFTVisualizer, use the following command steps:

1. set_system_mode analysis
2. open_visualizer

Then choose **Tools > Analyze DRC** from the DFTVisualizer menu and select the desired C4 DRC violation in the dialog box. The gates between the source cell and the failing cell, including the clock cone data for the failing clock, are displayed in the Debug window as shown in Figure 8-13.

Tip: Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example:

analyze_drc_violation c4-1

Note

In some situations, the tool's analysis may require significant CPU run time. You can interrupt the process and return to the command prompt using the Control-C key (intermediate results are not retained if you interrupt the analysis), or you can display periodic progress using the -Interval switch with the set_drc_handling command.

How to Debug from the Tool Command Line

To view the location of the first occurrence of a C4 DRC violation from the tool command line, use the following example command steps:

1. set_system_mode analysis
2. report_drc_rules C4-*occurrence#*
3. set_gate_report Clock_cone *pin_name*
4. report_gates *source_gate_id sink_gate_id*

The following transcript excerpt shows an example of the use of this command sequence starting at step 2:

```
report_drc_rules c4-1
// Warning: Clock /CLK failed rule C4 on input 3 of /DFF2 (10). (C4-1)
// Source of violation: input 3 of /DFF1 (9)
set_gate_report clock_cone /CLK
report_gates /DFF1 /DFF2
//  /DFF1 dff
//      clk    I  (C)/CLK
//      d      I  (-)  1-
//      q      O  (E)  5-/DFF2/d /NAND1/i0
//  /DFF2 dff
//      clk    I  (B)  7-/NAND1/out
//      d      I  (-)  5-/DFF1/q
//      q      O  (-)  6-/q2
```

You can see that the DRC violation is occurring because the clock input of /DFF2 is in both the clock and effect cones of /CLK.

Possible Resolutions

Note

There are special cases where C4 rule violations do not cause problems and can be ignored. These cases are rare, however, so be sure you understand how your circuit behaves versus what the tool simulates before deciding to ignore C4s.

There are two methods you can use to assure C4 rule violations do not result in simulation mismatches:

- Split Capture Cycle

  The first and preferred method is to enable the set_split_capture_cycle command (the tool enables it automatically if you use create_patterns). This forces a second evaluation of the cells affected by C4 DRC violations. The additional evaluation allows the new data to propagate and in turn be captured into the sink gate. If additional evaluation is required, a side effect of this command is an increase in simulation run time. If the increase in run time is excessive, you may want to use the other method, capture handling.

Note

When you use "set_split_capture_cycle on," you will still get C4 rule violations but you can ignore them.

- Capture Handling

  The second method of addressing C4 DRC violations is to use the set_capture_handling command. A side effect of this command is reduced test coverage if the number of C4s is significantly high. The item to consider is that only one simulation value is used for a given source. As a result, the sink gates can only capture either new or old data. Given the case where a source drives two sinks and one sink captures old data and the other sink captures new data, one of the sinks must capture X. The advantage of set_capture_handling is there is no impact on ATPG run time.

Mentor Graphics recommends you use "set_split_capture_cycle on." In cases where the impact on run time is severe and the number of C4 violations is small, then the use of capture handling is recommended. Do not use both methods simultaneously, however.

Screening Out False C4 Violations

For performance reasons, the tool's default DRC may occasionally report a false C4 violation. There are three additional analyses you can do to screen out these false C4s:

- Use the set_drc_handling command with the Atpg_analysis option. This option causes DRC to additionally check the clocks of the source and sink to see if they are gated off.
- For LSSD based designs, use the -Mode A option to the set_drc_handling command.

  When you specify this option for a selected clock, the rules checker evaluates all latches associated with the specified clock and categorizes their clock ports. It then uses these categories to determine if a violation exists. The following list describes each of the clock port categories:

  - **Inactive low (IL)**
    When the selected clock is low, the clock port of the latch is inactive.

- **Inactive high (IH)**
  When the selected clock is high, the clock port of the latch is inactive.
- **Active high slave (AHS)**
  When the selected clock is high, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.
- **Active low slave (ALS)**
  When the selected clock is low, the clock port of the latch is active. The data line of this latch connects (through buffers and inverters) to another latch called the data latch. When the clock port of the latch is active, all clock inputs of the data latch must be inactive. When the clock port of the latch is inactive, at least one clock input of the data latch must be active. Finally, non-clock primary inputs must not affect the clock inputs of the data latch.

During this evaluation, the rules checker prints a summary message that identifies the number of latches with clock ports placed in each category. If you enable learn reporting with “set_learn_report on,” you can then use report_gates to report on the individual latches in these categories.

You can screen out false violations of the C4 rule by issuing the set_drc_handling command before rules checking. The command usage in this context is:

set_drc_handling C4 [-Mode A *clock_name*]

The tool ignores violations of the C4 rule if the following conditions are true:

- The source latch port is IL or AHS and all paths from the source latch to the failing latch are blocked when the selected clock is high.
- The failing latch port is IH or ALS and all paths from the source latch to the failing latch are blocked when the selected clock is low.
- The violation source clock input is high and all other clock, set, and reset inputs are low for the source latch.

Related Commands

add_capture_handling
report_gates
set_capture_handling
set_drc_handling
set_gate_report
set_learn_report
set_split_capture_cycle

C5

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A clock pin must not be capable of simultaneously capturing data on multiple ports of the same scannable memory element. The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained value, and initialized non-scan cells are set to their stable state.

The rule violation occurs on a clock pin when multiple clock inputs of a scannable memory element are in the same clock cone and the clock inputs may be on at the same time. The tool performs a mutual exclusivity check to determine if both clock inputs associated with the failure can be active at the same time. If the justification results in a conflict without justifying decision nodes, it will not be considered a rules violation.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a race condition that creates inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Clock P failed rule C5 on input I of N (G). (C5-1)
```

P is the pin name of the clock, C5 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

The summary message is:

```
There were N clock rule C5 fails (clock is connected to multiple ports of
same  latch).
```

N is the number of occurrences of rules violation C5.

C5 Rule Violation Example

Figure 8-15 shows an example circuit and circuit setup specified in the tool.

Figure 8-15. C5 Rule Example Circuit

add_clocks 0 CLK
add_clocks 0 RST

If you run rules checking on this design given the setup commands shown, you will get a C5 rules violation. In this design, the RST signal connects to both the PRE and CLR pins of the second flip-flop. If you examine the inputs, you should see that A and NOTA should always have opposite values. If the tool knows this information, it knows that only one of the CLR or PRE signals can be active at any given time. To specify this information and fix the C5 rules violation, add the command:

add_input_constraints a -inv nota

C6

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A clock must not affect data that it is capturing. If it does, a race condition may result that produces inaccurate simulation results.

The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the clock cone are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a clock input of a scannable memory element and its data line are in the same clock cone.

The default handling for this rule violation is warning. When a violation occurs, you can access the clock cone data by the command "set_gate_report drc C6-N," where N is the c6 fail occurrence number, then issuing the report_gates command for the gate ID number displayed in the occurrence message. This identifies the problem input. By tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and "-" indicates no cone.

Note

Default C6 checking may miss some C6 violations when "set_clock_restriction off" is used. A more conservative analysis that identifies those C6 violations can be enabled by "set_drc_handling c6 -CONSERvative on." However, we generally recommend not to turn off clock restriction so that the -conservative switch does not need to be used.

The occurrence message is:

```
Clock P failed rule C6 on input I of N (G). (C6-1)
```

P is the pin name of the clock, C6 is the rule ID number, I is the gate input number of the clock line, N is the instance name of the gate, and G is the gate ID number.

The summary message is:

```
There were N clock rule C6 fails (clock may capture data affected by
itself).
```

N is the number of times a C6 rule violation occurred.

C6 Rule Violation Example

Figure 8-16 shows an example circuit and circuit setup specified in the tool.

Figure 8-16. C6 Rule Example Circuit

If you ran rules checking on this design, given the setup commands shown, you would get a C6 rules violation. The default handling for this rule violation is warning. In this design, the CLK signal goes to both the CLK and D inputs of the first flip-flop. Thus, data can be captured in this flip-flop that may be affected by the capturing clock. To prevent the CLK signal from influencing the data, add the command:

add_input_constraints -c0 SC_EN

Constraining the SC_EN signal to 0 ensures that changes in the clock will not change the data.

For ATPG, you can handle C6 rule violations by issuing a “set_clock_off_simulation on” command prior to creating patterns. See the set_clock_off_simulation command description for additional information.

In some designs, different flip-flops may capture the off state and on state of the clock causing the simulation to fail, In this case, you can use one of the following two commands to mask the signal creating the C6 violation. Note the add_cell_constraints command is more restrictive than the set_simulation option command.

add_cell_constraints -Drc C6

set_simulation_options -C6_mask_races ON

For more information, see the add_cell_constraints and set_simulation_options commands.

C7

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Each clock input (not including set and reset lines) of a scan or non-scan cell memory element must be capable of capturing data when a single clock primary input line is on and all other clocks are off. It is acceptable that this may require placing values on non-clock primary inputs or scan cells. The application performs this check using the simulated values that result when one clock is set to X, all other defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states. The rule violation occurs when a clock input of a scan cell always remains off.

The default handling for this rule violation is warning. Failure to satisfy this rule indicates a scan cell clock input cannot capture data, resulting in some loss of test coverage.

The occurrence message is:

```
Clock input I of N (G) cannot capture data with a single  clock on. (C7-1)
```

I is the input number, N is the instance name, G is its gate index number, and C7 is the rule ID number.

The summary message is:

```
There were N clock rule C7 fails (scan cell capture ability  check).
```

N is the number of occurrences of rules violation C7.

C7 Rule Violation Example

Figure 8-17 shows an example circuit and circuit setup specified in the tool.

Figure 8-17. C7 Rule Example Circuit

```
add_clocks 1 PRE CLR
add_clocks 0 SCAN_CLK
add_input_constraints -c0 SCAN_MODE
```

If you run rules checking on this design given the setup commands shown, you will get a C7 rules violation. This type of error commonly occurs when incorrect clock or set/reset gating occurs in the design. This design constrains the SCAN_MODE signal to a constant 0 during ATPG. This constraint prevents the first flip-flop from ever being clocked, and from ever capturing data. To fix this problem, delete the pin constraint:

```
delete_input_constraints -all
```

C8

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

You may not directly connect a clock to a primary output (PO). The application performs this check by determining the forward cone of influence for a clock pin (clock cone). The bounds for the cone of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values, and initialized non-scan cells are set to their stable states. The rule violation occurs when a primary output is in the clock cone.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in the occasional usage of a different type of scan pattern, in which the tool observes only the POs directly connected to clocks. There will be no loss of test coverage or risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
Primary output P is connected to clock C. (C8-1)
```

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

```
There were N clock rule C8 fails (PO connected to a clock  line).
```

N is the number of occurrences of rules violation C8.

C8 Rule Violation Example

Figure 8-18 shows an example circuit and circuit setup specified in the tool.

Figure 8-18. C8 Rule Example Circuit

```
add_clocks 1 PRE CLR
add_clocks 0 CLK
```

If you run rules checking on this design given the setup commands shown, you will get two C8 rules violation. This type of violation is not usually a problem.

In this example, the connection of the PRE and CLR lines (which the tool considers clock lines) to the design's primary outputs causes the violations. The only way to get rid of these violations is to deliberately hold CLR and PRE off, but this would result in a number of ATPG untestable faults. A more acceptable solution is to accept this warning, or turn it off with the command:

set_drc_handling c8 ignore

C9

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Data captured by any clock with a direct path (through combinational logic only) to a primary output must not affect the direct path to a primary output of that same clock. The application performs this check by determining the forward cone of influence for a clock pin (clock cone) and for each scannable memory element influenced by the clock pin (effect cone). The bounds for the cones of influence are scan cells and circuitry set to a fixed value when constrained pins are set to their constrained values and initialized non-scan cells are set to their stable states. The rule violation occurs on a clock pin when a primary output is in both the clock cone and the effect cone.

The default handling for this rule violation is warning. Failure to satisfy this rule may result in a small loss in test coverage. There will be no risk of inaccurate simulation results. When an error condition occurs, you can access the cone data by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the problem input, and by tracing back from this input, you can identify how to correct the problem. C indicates clock cone, E indicates effect cone, B indicates both, and “-” indicates no cone.

The occurrence message is:

```
PO P path from clock C is gated by scan cell that uses same  clock. (C9-1)
```

P is the pin name of the primary output and C is the pin name of the clock.

The summary message is:

```
There were N clock rule C9 fails (PO connected to a clock  line gated by
scan  cell that uses same clock).
```

N is the number of occurrences of rules violation C9.

C9 Rule Violation Example

Figure 8-19 shows an example circuit and circuit setup specified in the tool.

Figure 8-19. C9 Rule Example Circuit

```
add_clocks 1 PRE CLR
add_clocks 0 CLK
```

If you run rules checking on this design given the setup commands shown, you will get a C9 rule violation. This type of violation is not usually a problem. However, a C9 violation can result in reduced coverage because it may introduce sequential effects into the generated clock patterns. In this case, the violation is at the SC_OUT line. The PRE signal can affect the data of the flip-flop and the gate at the output of the scan cell. The only way to get rid of this violation is to deliberately hold PRE off, but this would result in a number of ATPG untestable faults. A more acceptable solution is to accept this warning, or turn it off with the command:

set_drc_handling c9 ignore

C10

Category: Clock

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A sequential element (latch or flip-flop) can only be clocked once in any one pattern cycle. the tool will only pulse a clock primary input once in each cycle, or apply a clock procedure only once, therefore a failure of this rule implies that the circuit generates two or more internal pulses from a single clock pulse or clock procedure.

The default handling of this rule violation is error. Failure to satisfy this rule will result in creation of patterns which are likely to be incorrect and to fail both in verification and on the tester.

The occurrence message is:

```
Cell c might capture more than once by applying clock ck. (C10-1)
```

Figure 8-20 shows an example circuit which will generate a C10 error.

Figure 8-20. C10 Rule Example Circuit

In order to remove the violation, the circuit must be modified. Test logic can be added to block the path from one pulse generator to the OR gate during test mode.

Note

It is also possible to violate this rule by OR-ing together two clocks which are pulsed at different times in a clock procedure.

C16

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

When switching from the load_unload phase to the capture phase, sometimes the loading values of scan cells can be disturbed due to pin constraints specified in the capture phase. The C16 clock rule checks for this situation and issues a violation if it occurs.

For example, consider a DFF scan cell with two clock ports: the first port is used for scan loading and the second is used for normal operation. Assume the second port's clock signal (whose off state is defined as 1) is ANDed with an enable signal to produce the actual clock input value. If in the load_unload procedure the enable is forced to 0, the load value cannot be disturbed by the second clock port during scan loading. However, if you issued an add_input_constraints command in setup mode to set the enable signal to 1, the scan load value can be disturbed at the start of the capture cycle when switching from load_unload to capture. This is because the clock input at the second port will change from 0 to 1 when the enable changes from 0 to 1 as a result of the pin constraint.

Note

The C16 rule check will ignore an X at a scan cell's clock port, as it is possible to assign a proper value at the PIs to keep the scan cell stable at the beginning of the capture phase. Xs on scan cell clock ports will, however, cause C1 violations.

The default handling for rule C16 is error. Failure to satisfy this rule may result in simulation mismatches during verification.

To debug a C16 rule violation, issue a "set_gate_report error_pattern" command, then use report_gates for the gate instance named in the C16 occurrence message.

Caution

If you choose to ignore C16 violations (not recommended), you should add a CX cell constraint to each scan cell where a violation occurred—in order to avoid simulation mismatches. Be aware that each cell you constrain in this manner reduces test coverage. You can obtain the instance name of the scan cell from the occurrence message and use the add_cell_constraints command to apply the constraint to the cell's output pin. For example:

report drc rule c16

// Warning: The load value of scan cell /dff2/dff1/Q_reg (47) is disturbed by clock input 3 at the beginning of the capture phase when pin constraints are forced. (C16-1)

add_cell_constraints /dff2/dff1/Q_reg/Q CX

The occurrence message is:

```
The load value of scan cell N is disturbed by clock input I at the
beginning of the capture phase when pin constraints are forced. (C16-1)
```

N is the instance name of the gate, I is the gate input number of the clock line, and C16 is the rule ID number.

The summary message is:

```
There were N clock rule C16 fails (scan cell load value disturbed at
beginning of capture phase when pin constraints are forced).
```

N is the number of occurrences of rules violation C16.

C17

Category: Clock

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note for patterns -scan context.Warning for dft -scan and dft -test_points contexts.

report_drc_rules: Supported

Description

The default handling for the C17 rule in patterns -scan context is Note. When handling is set to Note, by default, all internally-driven glitch sources in a design are identified (as C17 violations), but they are not remodeled and will maintain a constant value during DRC and pattern generation.

The default handling for the C17 rule in the dft -scan and dft -test_points contexts is Warning. When handling is set to Warning, all internally-driven glitches are identified (as C17 violations) and are also remodeled as possible X sources during DRC and pattern generation.

You can change the DRC handing for C17 in any of the supported contexts. In addition to setting handling to Note and Warning, you can also set handling to Error or Ignore. Setting handling to Error reports the first violation and then aborts the system mode transition. Setting the handling to Ignore bypasses the C17 rule completely.

An internally driven glitch source consists of either a two-input AND gate, or a two-input OR gate where both inputs are driven from a common source (either directly or through buffers and inverters), such that one path is inverting and the other path is not. This type of circuit maintains the same constant output value regardless of the state of the driving node, but the output will probably contain a glitch when the driving node changes state.

With this rule, gates are identified as internally-driven glitch sources if they are both driven by a sequential element and also drive the clock or set/reset line of a sequential element.

Note

Internally-driven is defined as a glitch generator driven from a sequential element, and not from a primary output.

Remodeling these internally-driven glitch sources as a constant value (TIE0 or TIE1) may cause failures during scan chain tracing. For this reason, if handling is set to Warning, C17 identifies these internally-driven glitch sources and remodels them as an X source (TIEX gate).

You can use the report_drc_rules command to display a list of the internally-driven glitch sources identified in your design.

C19

Category: Clock

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The C19 rule checks any clock gating structure where a leading edge (LE) or trailing edge (TE) flop is gating a clock path that drives any state elements that can capture a value at the same edge.

Note that the violation can lead to simulation mismatch due to a race condition between the clock enable path and the data ports of the controlled state elements. You should investigate and fix C19 violations by changing the design or adding constraints to prevent the clock enable signal from changing the value while the data ports of the controlled state elements are capturing data.

To debug a C19 rule violation, issue the analyze_drc_violation command to display the following in the Debug window:

- path from the clock enable flop to the driving flop with the violation
- path from clock pin to enable flop
- constraint value
- clock enable flop and driving flop with the violation

Example of C19 Violation

The following example shows a design with 10 C19 violations:

SETUP> **set_system_mode analysis**
…

// 10 sequential elements are gating clock path and driving sequential elements on the same clock
// edge which can create a race condition. (C19)
…

ANALYSIS> **report_drc_rules c19-2**

// Warning: Active-high latch /Ibasic_gates/Igated_C3/ (30365) is gating clock pin /clk_3 (17) and
// driving some leading-edge flops which can create a race condition. (C19-2)

// The first traced sequential element driven by /Ibasic_gates/Igated_C3/ (30365) that can create a
// race condition is /Ibasic_gates/Iflop2o_ne_p23s_o_2S3/ (30334).

ANALYSIS> **report_drc_rules**
…

C19: #fails=10 handling=warning/atpg (clock gater race condition check)
…

ANALYSIS> **analyze_drc_violation C19-1**

This command produces the following display in the Debug Window:

Scan Cell Data Rules (D Rules)

The DFT tools check scan cells to ensure they are able to properly control and observe their data. You may select the handling of these scan cell data rules to be error, warning, note, or ignore. The following subsections describe these rules.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

D1

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Checks for the possible disturbance of data values loaded or captured into scan cells.

During the application of the test procedures, if other circuitry disturbs the data values loaded or captured into scan cells, those cells cannot be controlled or observed. The tool performs this check using the simulated values of each time period of the test procedures. A violation occurs if any clock input (including set and reset lines) of any scan cell allows data capture at an inappropriate time, in which case the control value loaded or the capture value unloaded from the scan cell cannot be trusted. Common sources of D1 violations are undefined clocks, problems in the test procedure file, or possibly design errors.

Effect on Testability

Failure to satisfy this rule can result in simulation mismatches when you verify the test patterns in a timing-based simulator. If the violations occur during the shift procedure, you may see related simulation mismatches in the serial pattern test bench only, not the parallel test bench.

If the number of D1 violations is small compared to the number of scan cells in the design, you may choose to handle this rule violation as a warning by issuing the set_drc_handling D1 Warning command prior to DRC. When “set_drc_handling d1” warning is in effect for DRC, the tool automatically applies an XX cell constraint to each D1 failing scan cell. This prevents the mismatches, but may reduce test coverage slightly. See the add_cell_constraints command description for information about the XX cell constraint.

Note

If “set_drc_handling d1” note is in effect for DRC, the tool will *not* apply any XX cell constraints for D1 violations, which ensures test coverage is not affected. However, you should use “note” handling only for cell architectures you know will not produce mismatches.

If the number of D1 violations is large, you should isolate and resolve the source of the failure(s) to avoid a major impact on test coverage.

How to Debug D1 Violations

The occurrence message is:

```
// N (G) disturbed during time T of P procedure. (D1-1)
```

N is the instance name of the scan cell memory element, G is the gate ID number, T is the time period, and P is the group test procedure name.

The summary message is:

```
// There were N occurrences of scan cell disturbs. (D1)
```

N is the number of occurrences of rules violation D1.

Use the report_drc_rules command to obtain additional information about the violations. For example, to view the occurrence messages for all D1s, use:

report_drc_rules d1

The occurrence messages list gate names and gate IDs you can copy and paste into commands during later debugging. You can report on a specific occurrence by issuing the command with "D1-" and the occurrence number. For example:

report_drc_rules d1-1

```
//  Error: /dataout/reg_q_0_ (373) disturbed during time 0 of grp1
           load_unload procedure. (D1-1)
```

When a D1 error occurs, you can access the simulated values at the gate where the error occurred by issuing the command, set_gate_report Error_pattern, then using the report_gates command to report the gate whose ID number is displayed in the error message. In the resultant display, you can identify the clock input not held at its off state. By tracing back from this input, you can usually identify how to correct the problem.

You can debug a specific occurrence of a D1 violation using DFTVisualizer or by issuing commands from the tool's command line. Examples of both methods follow.

How to Debug with DFTVisualizer

To view the location of a D1 violation using DFTVisualizer, use the following command steps:

1. open_visualizer
2. set_gate_level Primitive

Then choose **Tools > Analyze DRC** from the DFTVisualizer menu, select the ID of the desired D1 violation occurrence in the dialog box and click Analyze. DFTVisualizer will display the scan cell affected by that occurrence.

Tip: Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example, to display the first occurrence of a D1 violation:

analyze_drc_violation d1-1

Once the offending scan cell is displayed, trace back from the cell's clock input to the primary input that drives the clock. Ensure the clock's off state, which you defined using the add_clocks command, is correct.

How to Debug from the Tool Command Line

To view the location of a D1 violation from the command line, use the following example command steps:

1. set_gate_report Error_pattern
2. set_gate_level Primitive
3. report_drc_rules D1-*occurrence#*
4. report_gates *offending_gate's_id#*

The following transcript excerpt shows an example of the use of this command sequence:

```
set_gate_report error_pattern
set_gate_level primitive
report_drc_rules d1-1
//  ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//         load_unload procedure. (D1-1)
report_gates 373
//  /dataot/reg_q_0_ (373)  DFF
//      "S"    I  (0)  43-
//      R      I  (0)  157-
//      CLK    I  (1)  313-/dataot/ix104/Y
//      "D0"   I  (X)  255-
//      "OUT"  O  (X)  127-  128-
```

You would then trace back from the CLK input to try to determine why it is in an active state, which disturbed the cell's value.

In some of the more complex cases, the problem can be a clock disturb early in the capture cycle. In these cases, the error report does not provide enough detail to isolate the failure. To get more information, you can use the following command steps:

1. set_gate_report Drc_pattern State_stability
2. set_system_mode analysis

3. report_gates *offending_gate's_id#*

The following transcript excerpt shows an example of the use of this command sequence:

```
set_gate_report drc_pattern state_stability
//  Creating schematic for 3 instances (0 were compacted).
set_system_mode analysis
//  ---------------------------------------------------------------------
//  Begin scan chain identification process, memory elements = 56.
//  ---------------------------------------------------------------------
//  Reading group test procedure file fast.testproc.
//  WARNING: Pin tclk is used for both write control and pulse clock
//  Simulating load/unload procedure in grp1 test procedure file.
//  Chain = chain1 successfully traced with scan_cells = 12.
//  Chain = chain4 successfully traced with scan_cells = 12.
//  Chain = chain3 successfully traced with scan_cells = 12.
//  Chain = chain2 successfully traced with scan_cells = 12.
//  48 scan cells have been identified in 4 scan chains.
//  Longest scan chain has 12 scan cells.
//  WARNING: 8 edge-triggered clock ports set to stable high. (D7)
//  ERROR: /dataot/reg_q_0_ (373) disturbed during time 0 of grp1
//         load_unload procedure. (D1-1)
//  ERROR: Rules checking unsuccessful, cannot exit SETUP mode.
report_gates 373
//  /dataot/reg_q_0_ (373)  DFF
//                (ts)(ld)(shift)(cap)(stbl)
//      "S"    I  ( 0)( 0)(000~0)(000)(   0)   43-
//      R      I  ( 0)( 0)(000~0)(XXX)(   X)   157-
//      CLK    I  ( 1)( 1)(101~1)(XXX)(   X)   313-  /dataot/ix104/Y
//      "D0"   I  ( X)( X)(XXX~X)(XXX)(   X)   255-
//      "OUT"  O  ( X)( X)(XXX~X)(XXX)(   X)   127-  128-
```

The item to notice is the clock values during the capture cycle. In this case the value of the clock is (XXX). This indicates the clock is not at its off state at the beginning or end of the capture cycle. The correct values should be either:

- (0X0)—For a leading edge device with an off state of 0 for the clock, or
- (1X1)—For a trailing edge device with an off state of 1 for the clock

Possible Resolutions

If your debugging effort shows an offending clock's off state is incorrect, use the add_clocks command to redefine the clock with the correct off state. To change the defined off state, first delete the old definition with the delete_clocks command, then reissue add_clocks. To see a clock's current defined off state, use the report_clocks command.

If the clocks are working correctly, examine the test procedure file. By tracing through the design, you may find an incorrect constraint or force value that can be fixed in the test procedure file.

Note

If you make any changes in the test procedure file, execute the read_procfile command with the correct filename before re-checking the design rule checks.

Related Commands

add_clocks
analyze_drc_violation
delete_clocks
report_clocks
report_drc_rules
report_gates
set_drc_handling
set_gate_report

D2

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

During system data capture, if the scan path from a MASTER or SLAVE element to its COPY is sensitized, it must be unique. The tool performs this check by comparing the inputs of the COPY with its associated memory element. A violation occurs if there are multiple paths from the associated memory element (MASTER or SLAVE) to the COPY and these paths can be sensitized at the same time. The application checks for the following conditions:

- The value on the data line of the COPY element must be able to be propagated back to its associated memory element along the scan path when constrained pins are set.
- The scan path, when sensitized from the associated memory element to the COPY, must be unique.
- The COPY and its associated memory element may have only a single clock port.
- For a COPY and its associated memory element, all non-tied clock, set, and reset inputs must have the same or equivalent source.

Note

DRC does not consider it a D2 violation if a COPY element can capture a value through a different path other than from its associated memory element during system capture.

Failure to satisfy this rule usually reduces test coverage. You can avoid the coverage loss by using the command, set_split_capture_cycle On.

The occurrence message is:

```
COPY N (G) failed data capture check. (D2-1)
```

N is the instance name of the COPY memory element, and G is the gate ID number.

The summary message is:

```
N COPY scan elements failed data capture check. (D2)
```

N is the number of occurrences of rules violation D2.

D3

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

For all scan cells that contain a SLAVE, the **master_observe** procedure must propagate the data value of the MASTER memory element to the SLAVE. The application performs this check using the simulated values of each time period of the **master_observe** procedure to trace back from the SLAVE to the MASTER. The rule violation occurs if the **master_observe** procedure does not properly sensitize the path between the SLAVE and MASTER.

The default handling for this rule violation is error. You may ignore this error condition by issuing the command set_drc_handling D3 Warning.

When the handling is set to other than error, the tool will automatically make the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERs, test coverage may be reduced.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the master_observe argument and the desired time) and using the report_gates command for the gates in the SLAVE to MASTER path. This identifies the location of the blockage; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
N (G) not successfully observed by master_observe  procedure. (D3-1)
```

N is the instance name of the MASTER memory element, and G is the gate ID number.

The summary message is:

```
N MASTERs not successfully observed by master_observe  procedure. (D3)
```

N is the number of occurrences of rules violation D3.

D4

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

If you define the **skew_load** procedure, it must propagate the data value of the preceding scan cell (or scan chain input pin) to a MASTER memory element. The application performs this check using the simulated values of each time period of the **skew_load** procedure to trace back from a MASTER to its preceding scan cell output or scan chain input pin. The rule violation occurs if the **skew_load** procedure does not properly sensitize the path.

The default handling for this rule violation is error. Failure to satisfy this rule may result in inaccurate simulation results when you use the skew load option. The **skew_load** procedure is optional and you can avoid rules violations by removing the procedure definition.

When an error condition occurs, you can access the simulated values by setting the gate reporting to drc_pattern (with the skew_load argument and the desired time) and using the report_gates command for the gates in the path. This identifies where the blockage occurred; by tracing back from the inputs, you can identify how to correct the problem.

The occurrence message is:

```
Skew_load procedure not successful for MASTER %N (G). (D4-1)
```

N is the instance name of the MASTER memory element and G is the gate ID number.

The summary message is:

```
Skew_load procedure not successful for N MASTERs. (D4)
```

N is the number of occurrences of rules violation D4.

D5

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

All memory elements (latches and flip-flops) must be scannable. The application performs this check after identifying all scan memory elements. The rule violation occurs for all memory elements not identified as part of a scan cell.

Note

The D5 check simulates the circuit as purely combinational by using a sequential depth of 0 or 1 (as if the "set_pattern_type -Sequential" command is set to 0 or 1).

When the tool identifies a non-scan memory element, the tool classifies each element into one of the following types:

- INIT-0: If it is at 0 at the beginning of the first capture cycle.
- INIT-1: If it is at 1 at the beginning of the first capture cycle.
- INIT-X: If its state is unknown at the beginning of the first capture cycle and it may go to any state during capture.
- TIE-0: If it is always at 0 during capture.
- TIE-1: If it is always at 1 during capture.
- TIE-X: If it is always at an unknown state during capture.
- TLA: If it is always transparent when its clock is at its off state.

The model used (and reported in the D5 violation message) is determined from the element's value in the D5 simulation, at the end of load_unload and before entering capture. For example, if the element's value is 1 at that time, the tool models the element as a TIE-1. Note, however, that the tool simulates the TIE-0, TIE-1, and TIE-X models as INIT-0, INIT-1, and INIT-X models, respectively. This means, for example, that a non-scan TIE-0 model can be evaluated after the beginning of the first capture cycle and get a value other than 0.

Latches modeled as TIE-X gates become candidates for transparent latches, sequential transparent cells, or clocked sequential cells if you set the pattern type appropriately with the set_pattern_type command.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in some loss of test coverage.

The occurrence message is:

```
N (G) is a non-scan T1 converted to T2. (D5-1)
```

N is the instance name of the non-scan memory element, G is the gate ID number, T1 is the gate type (latch or flip-flop), and T2 is the gate type that models it (TIEX, TIE0, or TIE1).

The summary message is:

```
N non-scan memory elements converted to T gates. (D5)
```

N is the number of occurrences of rules violation D5, and T is the gate type that models the non-scan cell. The tool displays a summary message for each remodeled gate type.

D6

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

All non-scan latches must behave as transparent latches. The application performs this check for all nonscan latches that are not set to a stable binary value. The rule violation occurs if a candidate latch fails one of the following conditions:

- If the latch creates a potential feedback path, that path must be broken by scan cells or non-scan cells other than transparent latches. For more information, refer to the set_tla_loop_handlingcommand.
- The latch must have a propagable path to an observable point.
- The latch must be capable of passing a value when all defined clocks are at their off-state.
- All clock, set, and reset inputs of the latch must either be set to a determinate state when all clocks are off and pin constraints are set, or must not connect to defined clocks.
- The latch must not have more than one set/reset/clock input on when all defined clocks are at their off-state.

Failure to satisfy this rule can reduce test coverage. The default handling for this rule violation is warning. If you set the handling for this rule to ignore, the tool will not perform this check and the design's latches will not be checked for transparency.

For scan insertion in the dft -scan context, if you want the tool to consider non-transparent latches as scan candidates, you must turn test logic on (with the set_test_logic command) and do one of two things: 1) set the handling of D6 to ignore, in which case the tool does not perform the transparency check and automatically considers the non-scannable latches for scan insertion; or 2) use the set_latch_handling Scan command, in which case the tool performs the check and considers non-transparent latches for scan insertion.

The occurrence message is:

```
Latch N (G) not transparent due to R. (D6-1)
```

N is the instance name of the non-scan latch, G is the gate ID number, R is the reason it cannot be transparent, and D6 is the rule ID number.

The summary message is:

```
N latches not transparent due to R. (D6)
```

N is the number of occurrences of rules violation D6, and R is the reason. The application displays a summary message for each reason.

D7

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

At the end of the **shift** procedure, the clock inputs of scan flip-flops must not be set to a one state. The application performs this check using the simulated values of the last time period of the **shift** procedure. The rule violation occurs if any clock input (not including set and reset lines) of any scan flip-flop (except COPY) is set to 1. A possible cause of a rules violation is an incorrect definition of the off-state of a clock.

Note

Some design practices consider this condition acceptable.

The default handling for this rule violation is warning. Failure to satisfy this rule will result in scan cells capturing data on the trailing edge of the capture clock pulse, thus resulting in some risk of race conditions.

The occurrence message is:

```
Flip-flop N (G) has clock port set to stable high. (D7-1)
```

N is the instance name of the non-scan memory element, G is the gate ID number, and D7 is the rule ID number.

The summary message is:

```
N edge-triggered clock ports set to stable high. (D7)
```

N is the number of occurrences of rules violation D7.

D8

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

If a MASTER latch only propagates to a SLAVE and can only capture data when the SLAVE is inactive, a clock input of the MASTER latch must not be active when all clocks are off. The system uses the **master_observe** procedure to observe the values placed into the scan cell and no longer considers the SLAVE to be observable.

The application performs this check using the simulated values that result when all defined clocks are at their off-state, the constrained pins are set to their constrained values, and the initialized non-scan cells are set to their stable states.

The rule violation occurs if a clock input of a MASTER latch is not off, the MASTER latch only propagates to a SLAVE, and can only capture data when the SLAVE is inactive.

Note

Some design practices consider this condition acceptable.

When an error condition occurs, you can access the simulated values by setting the gate reporting to error_pattern and using the report_gates command for the gate ID number displayed in the error message. This identifies the clock input not held off, and by tracing back from this input, you can identify how to correct the problem.

The default handling for this rule violation is error because for certain rare LSSD scan cell designs it can result in a simulation mismatch during pattern verification in a timing-based simulator. Failure to satisfy this rule will result in data captured into the MASTER without the application of a capture clock.

Note

If “set_drc_handling d8 warning” is in effect for DRC, the tool will automatically apply an SX cell constraint to each D8 failing master cell. This prevents the mismatches, but may reduce coverage. See the add_cell_constraints command description for information about the SX cell constraint.

If “set_drc_handling d8 note” is in effect for DRC, the tool will *not* apply any SX cell constraints for D8 violations, which ensures test coverage is not affected. However, you should use “note” handling only for cell architectures you know will not produce mismatches.

The occurrence message is:

```
MASTER latch N (G) allows data capture while clocks off. (D8-1)
```

N is the instance name of the non-scan memory element, and G is the gate ID number.

The summary message is:

```
Clocks at off-state allow data capture for N MASTER latches.  (D8)
```

N is the number of occurrences of rules violation D8.

D9

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Seq_transparent procedures must not disturb scan cells, primary outputs, or previously calculated seq_transparent cells. The default handling for this rule violation is warning. The application performs this check during simulation of the **seq_transparent** procedure.

A rule violation occurs under any of the following conditions:

- If scan cell state elements change during the application of a **seq_transparent** procedure. Unless these scan state elements capture new data at the application of the capture clock, the system cannot observe them during patterns that apply the procedure.
- If disturbed scan cells supply the inputs of other scan cell state elements. The system cannot observe these other scan cell state elements during patterns that apply the procedure.
- If a primary output comes from disturbed circuitry, the system cannot observe these primary outputs during patterns that apply the procedure.
- If the application of the **seq_transparent** procedure disturbs a non-scan state element that previously captured an undisturbed value.

Failure to satisfy this rule results in restrictions on the use of these points for observation during simulation and test generation for patterns that apply the procedure. This can reduce test coverage.

If a disturbed cell must support a valid seq_transparent cell, the tool identifies the disturbed cell as a seq_transparent cell and issues a violation of type seq_transparent cell disturb. Otherwise, the tool will not identify it as a seq_transparent cell, and will instead issue a D9 violation of type unused seq_transparent cell disturb.

The occurrence message is:

```
T disturb occurred on N (G) in procedure P. (D9-1)
```

T is the type of disturb, N is the instance name of the disturbed gate, G is the gate ID number, P is the name of the **seq_transparent** procedure, and D9 is the rule ID number.

The summary message is:

```
N T disturbs occurred in procedure P. (D9)
```

N is the number of occurrences of a disturbance type, T is the type of disturbance, P is the name of the **seq_transparent** procedure, and D9 is the rule ID number. The tool issues a summary message for each disturbance type.

D10

Category: Data

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported for patterns -scan context only.

Description

The transparent capture cells in **clock** procedures must not propagate both old and new data to other state elements. For example, a violation can occur when a clock procedure pulses two clocks, and a memory element clocked by the first applied clock feeds at least two other memory elements, clocked by each of the clocks. To illustrate this, assume the clock procedure is as follows:

```
procedure clock clock_proc1 =
   force A 1 1;
   force A 0 2;
   force B 1 3;
   force B 0 4;
end;
```

Given this procedure, the highlighted flip-flop in Figure 8-21, which gets old data from the first flip-flop when A pulses, violates this requirement.

Figure 8-21. Rule D10 Violation Example

The default handling for this rule violation is error. Failure to satisfy this rule will result in the source gate being modeled as a TIEX gate. You can suppress reporting the results of this check using set_drc_handling. However, regardless of how you set the handling, Tessent FastScan and Tessent TestKompress always perform this check on model violating gates with TIEX behavior.

The occurrence message is:

```
Cell N (G) has invalid transparency (X/Y) in procedure P. (D10-1)
```

N is the cell name, G is the gate ID number of the cell, X is the ID number of the gate capturing old data, Y is the ID number of the gate capturing new data, and P is the clock procedure name.

The summary message is:

```
There were N cells with invalid transparency.(D10)
```

N is the number of cells found to violate this rule.

D11

Category: Data

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

The transparent capture cells in **clock** procedures must not propagate data to primary outputs. For example, assume the clock procedure is as follows:

```
procedure clock clock_proc1 =
   force A 1 1;
   force A 0 2;
   force B 1 3;
   force B 0 4;
end;
```

Given this procedure, the primary output in Figure 8-22, which gets data from the first latch when A pulses, violates this requirement.

Figure 8-22. Rule D11 Violation Example

The default handling of this violation is warning. Failure to satisfy this rule results in the affected PO not being used for observation, as the expected value is always considered an X. Tessent FastScan and Tessent TestKompress classify faults detectable only through these POs as AU faults. To change the default handling, use the set_drc_handling command.

If you change the handling to ignore, Tessent FastScan and Tessent TestKompress do not perform this check and continue to use the primary outputs for observation. You may want to ignore this rule if you are running ATPG on a sub-block whose POs will eventually feed scan cells. In this case, a D10 violation may occur instead. If you ignore this violation, the reported fault coverage does not consider reconvergence through transparent capture cells, and ATPG could produce an invalid pattern set.

The occurrence message is:

```
Transparent_capture cell N (G) in procedure P has connectivity to
POs.(D11-1)
```

N is the cell name, G is the gate ID number of the cell, and P is the clock procedure name.

The summary message is:

```
There were N Transparent_capture cells with connectivity to POs.(D11)
```

N is the number of cells found to violate this rule.

D12

Category: Data

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

In the state stability analysis stage, DRC simulates a number of shift cycles to calculate the stable value (load value) of nonscan memory elements at the beginning of capture.

If some nonscan memory elements need more than one shift cycle to be initialized, and its load value is observed into a scan cell, the parallel simulation test bench may have simulation mismatches. This is because in parallel simulation test bench the tool generally only has one parallel shift cycle, leading to insufficient shift cycles to initialize those nonscan elements.

If state stability analysis takes more than one cycle to calculate nonscan load value, then the tool issues the following warning message:

```
//  Warning: DRC simulated N shift cycles to initialize some non-scan
//  memory elements to their load values. (D12-1)
//  Note: In order to avoid potential parallel simulation mismatches,
//  M post shift cycles will be automatically added to the testbench as
//  if the SIM_POST_SHIFT parameter keyword were set to M.
//  This will slow down parallel simulation. To avoid slowing down
//  parallel simulation, the number of cycles DRC simulates can be
//  restricted using "set_stability_check -max_shift_cycles 1". However,
//  initializing fewer non-scan memory elements may lead to lower test
//  coverage.
```

N is number of shift cycles and M is the number of shifts required.

Extra Rules (E Rules)

Extra rules identify potential design requirement problems. The following subsections describe the extra rules.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

E1

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

All scan cells must be LSSD scan cells that contain a master and a slave latch. They may also contain shadow latches. This rule is meant to enforce a strict LSSD architecture within a design. The application performs this check by inspecting the memory elements of all scan cells. The rule violation occurs when any memory element (that is not a latch or a scan cell) does not contain a SLAVE.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

```
MASTER N (G) is not an LSSD latch. (E1-1)
```

N is the instance name of the MASTER gate, and G is the gate ID number.

The summary message is:

```
N scan cells are not LSSD. (E1)
```

N is the number of occurrences of rules violation E1.

E2

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

There must be no data inversion between adjacent scan cells, the scan chain input pin (SCI) and its adjacent scan cell, and the scan chain output pin (SCO) and its adjacent scan cell. The application performs this check by inspecting the inversion data for all scan cells. The rule violation occurs when any adjacent scan cells (including SCI and SCO) have an inversion difference.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

```
Scan chain has inversion between N1 (G1) and N2 (G2). (E2-1)
```

N1 is the instance name of one MASTER (or SCI) gate, G1 is its gate ID number, N2 is the instance name of the other MASTER (or SCO) gate, and G2 is its gate ID number.

The summary message is:

```
There were N scan chain inversions. (E2)
```

N is the number of occurrences of rules violation E2.

E3

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

There must be no inversion between MASTER and SLAVE for any scan cell. The application performs this check by inspecting the inversion data for the memory elements of all scan cells. The rule violation occurs when the MASTER is inverted relative to its SLAVE.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no effect. The occurrence message is:

```
SLAVE N (G) is inverted relative to MASTER. (E3-1)
```

N is the instance name of the SLAVE, G is its gate ID number, and E3 is the rule ID number.

The summary message is:

```
There were N SLAVEs inverted relative to MASTER. (E3)
```

N is the number of occurrences of rules violation E3.

E4

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Checks for conflicting values driving the same net during evaluation of the test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the application of the test procedures. The tool performs this check using the simulated values of each time period of all test procedures except the capture procedure. A rule violation occurs if any bus gate is at an X state and two or more of its inputs are not at Z.

The default handling for this rule violation is warning.

Effect on Testability

Failure to satisfy this rule does not affect coverage, but will result in the risk of bus contention.

How to Debug E4 Violations

The occurrence message is:

```
Bus contention on N (G) occured at time T of P procedure. (E4-1)
```

N is the instance/net name of the bus gate, G is the gate ID number, T is the time period, and P is the procedure name.

The summary message is:

```
There were N occurences of bus contention in test procedures. (E4)
```

N is the number of times an E4 violation occurred.

You can access the simulated values at the gate where the error occurred by issuing the set_gate_report command with the Drc_pattern argument, then using the report_gates command to report the gate whose ID number is displayed in the error message. If this is the only DRC violation, or if you have changed the default handling to error, you can also view simulated values of the gate of interest by issuing set_gate_report with the Error_pattern argument prior to using report_gates. Gate reporting with the proper simulation data helps you identify the conflicting inputs; by tracing back from these inputs, you can identify how to correct the problem.

If you see an “A” appended to the “E4” in the occurrence message (E4-1-**A** for example), it indicates the occurrence is due to the tool aborting the E4 check before it was completed. As these are “possible” but unproven E4s, the tool may be able to screen out some of them if you raise the abort limit. Refer to “Possible Resolutions” on page 2237 for more information.

How to Debug with DFTVisualizer

To view the location of an E4 DRC violation using DFTVisualizer, use the following command steps:

1. set_gate_level Primitive
2. open_visualizer

Then choose **Analyze > DRC Violation** from the DFTVisualizer menu, select the violation ID of the desired E4 DRC violation in the dialog box and click Analyze. The tool will automatically change the gate reporting to “drc_pattern load_unload” and display the bus gate affected by that violation in the Debug window of DFTVisualizer.

Tip: Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* and -Display arguments at the tool’s command line. For example, to display the first occurrence of an E4 violation:

analyze_drc_violation e4-1 -display

Once the offending bus gate is displayed, trace backward from the bus gate. Figure 8-23 shows an example DFTVisualizer display of such a trace, where the starting point of the trace is the offending gate, /tsdo.

Figure 8-23. Example E4 Violation Trace in Debug Window of DFTVisualizer

How to Debug from the Tool Command Line

To view the location of the first occurrence of an E4 violation, use the following steps:

1. set_system_mode analysis
2. report_drc_rules E4-*occurrence#*
3. set_gate_level Primitive
4. set_gate_report Drc_pattern Load_unload
5. report_gates *gate_id*

The following transcript excerpts show, for the design segment of Figure 8-23, an example of the use of this command sequence starting at step 2:

```
report_drc_rules e4-1
// Warning: Bus contention on /tsdo/dout[0] (314) occurred at time 0 of
//          load_unload procedure. (E4-1)
set_gate_level primitive
set_gate_report drc_pattern load_unload
report_gates 314
//  /tsdo (314)  BUS
//      "I0"   I  (XX)  294-/tsdo/ix149/Y
//      "I1"   I  (XX)  286-/tsdo/ix145/Y
//      "OUT"  O  (XX)  396-/dout[0]  330-
```

The Xs on the two inputs to the bus gate in this example are the source of the problem. A reasonable way to proceed is to choose one input and trace back through the gates driving it until you find the source of the Xs. For example, you might first look at where the "I0" input comes from:

```
report_gates 294
//  /tsdo/ix149 (294)  TSD
//      E      I  (XX)  247-/ix310/Y
//      "D"    I  (XX)  221-
//      Y      O  (XX)  314-
```

Notice the Xs on the enable (E) input. This input is of particular interest because an X here results in an X on the TSD's output regardless of the value on its D input. So tracing back from the enable would be a logical next step:

```
report_gates -endpoints -backward 247
//  ---------------------------------------------------------------------
// Begin backward trace for gate /ix310 (247).
//  ---------------------------------------------------------------------
//  /tsden[1] (29)  PI
//      tsden[1]  O  (XX)  160-/ix292/A
//  /scan_en (32)  PI
//      scan_en  O  (11)  163-/ix140/A0  161-/ix138/A1  162-/senmux/ix9/A1
//  /e4 (34)  PI
//      e4     O  (XX)  163-/ix140/A1
//  Number gates in trace = 3.
```

If you could get a 0 on the net connected to this TSD's enable, it would remove the Xs on its output.

Possible Resolutions

Cell constraints and ATPG constraints (dynamic or static) have no impact on the E4 rule. Pin constraints can have an impact because the tool may add extra cycles at the end of the test_setup procedure if the procedure does not preserve the constrained value. The constrained value can be overridden by another force value in the test procedure.

It is common during test_setup, when a design is being initialized, to have valid but momentary bus contention before the design reaches its initialized state. The E4 rule often detects these momentary occurrences of bus contention and as a result, the tool may report many E4 violations for the test_setup procedure. If you know these momentary occurrences of bus contention are not a problem and do not wish to be reminded of them, you can disable the E4 checks for the test_setup procedure by issuing the command "set_drc_handling -Skip_procedure Test_setup."

Increasing the abort limit with the set_abort_limit command can help screen out false E4 violations by enabling the tool to perform more extensive contention checking. This works for E4A violations only.

E5

Category: Extra

Contexts Supported: dft -scan, dft -test_points, patterns -scan, patterns -scan_diagnosis

Default Handling: Note in patterns -scan contexts only. Ignore otherwise.

report_drc_rules: Supported

Description

When the application places constrained states on constrained pins and binary states on PIs and scan cells, X states must not propagate to an observable point. Failure to satisfy this rule will result in the risk of X states propagating to an observable point. This is a serious condition for BIST circuits, may reduce the effective compression obtainable with EDT, and has no effect on regular ATPG.

Note

Tessent TestKompress can handle the X states, but they typically increase the number of patterns generated, lowering the effective compression.

The application performs this check on gates that can create an X state with their inputs at binary values. It will not consider gates that do not have a path to an observable point, or that have all paths blocked by tied or constrained circuitry. The tool checks for the following conditions:

- A violation on a wired gate (WIRE) occurs if the tool can place different values on its inputs and the net resolution is set to wire.
- A violation on a BUS gate occurs if more than one of the BUS-connected tri-state drivers or switches turn on simultaneously, or all drivers turn off simultaneously and the Z state behaves as an X.
- A violation on a tri-state driver gate (TSD) or a switch gate (SW) occurs if it does not connect to a BUS gate. You can turn off the enable line, and the Z state behaves as an X.
- A violation on a TIE-X gate occurs if the gate is locally sensitizable up to the point where it has multiple fanouts (or observable points).
- A violation on a transparent latch (TLS) occurs if a single clock line is not set to its on-state, or the set and reset lines are not off.
- A violation on a ROM or RAM gate occurs if you can set a read line to off, the read_off value is X, and an output is sensitizable when the read line is off.
- A RAM/ROM violation also occurs if any memory element is uninitialized and an output is sensitizable to an observation point.

The default handling for this rule violation in patterns -scan context is note; in other tools, it is ignore. When a violation occurs, you can access the tied/constrained simulated values by setting

the gate reporting to “constrain_value” and using the report_gates command for the gate ID number displayed in the DRC message. The tool needs to be in a non-setup system mode. By tracing back and forward from this gate, you can identify why the violation occurred.

The occurrence message is:

```
T gate N (G) may have an observable X-state. (E5-1)
```

T is the gate type, N is the instance/net name of the gate, and G is the gate ID number.

The summary message is:

```
N gates may have an observable X-state. (E5)
```

N is the number of occurrences of rules violation E5.

E6

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

When the tool places constrained states on constrained pins, the inputs of a gate must not have sensitizable connectivity to more than one memory element of a scan cell. The tools consider a scan cell to have multiple memory elements if the cell contains, for example, a shadow element or copy element (refer to Slave Element and Shadow Element in the *Tessent Scan and ATPG User's Manual* for detailed information about these memory elements and their use in scan cells).

The application performs this check by tracing the forward cones of influence of all scan cell memory elements through unconstrained and untied circuitry. The rule violation occurs when any gate is in the cone of influence of more than one memory element of a single scan cell.

The default handling for this rule violation is ignore. Failure to satisfy this rule may result in some loss of test coverage, but most faults should be detectable using a skewed load test procedure.

The occurrence message is:

```
Multiple memory elements of scan cell P (C) are connected to N (G). (E6-1)
```

P is the position number of the scan cell, C is the chain name, N is the instance name of the gate, and G is its gate ID number.

The summary message is:

```
There were N scan cells with multiple memory element  connectivity to a
gate.  (E6)
```

N is the number of occurrences of rules violation E6.

E7

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

External bidirectional drivers must be at the high impedance (Z) state during the application of the test procedures. You can use this rule to ensure that no bus contention can occur at bidirectional pins independent of the force values on the bidirectional pins. The application performs this check using the simulated values of each time period of all test procedures (except **test_setup**). The rule violation occurs if any bidirectional tri-state driver is not at a Z state. Using the -Mode option with the set_drc_handling command and the report_drc_rules command, you can check the value being forced on the bidirectional pins.

The default handling for this rule violation is ignore. If rule E4 (which ensures bus-mutual exclusivity) passes, a violation of rule E7 has no effect. Failure to satisfy this rule (normally) has no effect if there is no violation of rule E4, which ensures no bus contention actually occurs.

You can access simulated gate values using the report_gates command with the gate data set to DRC_pattern. If this is the only DRC violation or if you have changed the default handling to error, you can also view simulated values using report_gates with gate reporting set to error_pattern. Gate reporting with the proper simulation data helps you identify the bidirectional pin that failed, and by tracing connectivity from this point, you can identify how to correct the problem.

Note

You should set the gate data prior to any violation analysis, to ensure the gate data reported is for the correct violation.

The occurrence message is:

```
BIDI pin P not set to input mode at time T of P procedure.  (E7-1)
```

P is the pin name of the bidirectional pin, T is the time period, and P is the procedure name.

The summary message is:

```
There were N occurrences of BIDIs not set to input mode during scanning.
(E7)
```

N is the number of occurrences of rules violation E7.

E8

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

All scan cell MASTER elements of a scan chain must use a single shift clock. If the scan cells contain slaves, all slaves of all scan cells of a scan chain must also use a single shift clock. You can use this rule to ensure that the tester does not cause clock skew problems during the loading and unloading of the scan chains. The application performs this check by inspecting the memory elements for all scan cells of a chain. The rule violation occurs when a chain uses multiple clocks to shift master or slave data.

If multiple shift clocks pulse in the shift procedure and they are blocked from reaching the scan chain by the effects of pin constraints, you must specify the set_drc_handling command with the Atpg_analysis option to avoid an error. This checking considers only pin constraints that have not been overridden by force statements during the shift and load_unload procedures.

The default handling for this rule violation is ignore. Failure to satisfy this rule will have no ATPG effect. The occurrence message is:

```
Multiple clocks were used to shift T of scan chain C. (E8-1)
```

T is the type of scan cell memory element (MASTERs or SLAVEs), and C is the chain name.

The summary message is:

```
There were N occurrences of multiply clocked scan chains.  (E8)
```

N is the number of occurrences of rules violation E8.

Note

If you set the DRC E8 handling to anything other than Ignore, the report_scan_chains command additionally displays the clock of each scan cell. This enhanced report is only available in non-Setup mode.

E9

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

The drivers of wire gates must not be capable of driving opposing binary values. The application performs this check by attempting to satisfy the placement of opposing binary values on all combinations of the two drivers of a wire gate. The rule violation occurs at a wire gate if it is possible to satisfy those conditions for at least one combination of drivers. When a violation occurs, the tool identifies the failing wire gate and the drivers capable of being placed at opposing values.

This rule ensures that there is no possible contention (for the good machine) on wire gates. The tool will not perform this rule check on wire gates whose behavior you changed to AND or OR using the set_net_resolution command. Also, the tool does not consider pin constraints and equivalences with this check.

The default handling for a violation of this rule is set to ignore. A violation of this rule indicates the possibility that patterns exist that have contention on wire gates. The occurrence message is:

```
WIRE gate N (G) has possible contention on drivers G1 and G2. (E9-1)
```

N is the gate name of the wire gate, G is its gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N WIRE gates which may have possible contention.  (E9)
```

N is the number of occurrences of rules violation E9.

E10

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Checks for bus contention mutual-exclusivity during measure_po events in test procedures.

Tri-state drivers must not have conflicting values when driving the same net during the measurement of primary output (PO) values (measure_po events in test procedures). This rule differs from the E4 rule in that it is not checked during test procedure events other than measure_po events. The tool performs this check by analyzing each dominant strong bus to determine if it can cause contention. The analysis places each bus in one of the following categories:

- **Pass** - Test generation analysis determines a contention condition cannot occur.
- **Fail** - Test generation analysis identifies a possible contention condition.
- **Abort** - Test generation analysis terminated while attempting to determine if a contention condition could occur.
- **Bidi** - Test generation determines that the bidirectional pin (which can only have a single tri-state driver) can potentially create a contention condition.

Buses in either the fail or abort categories violate this rule. Buses in the bidi category require checking during post-DRC ATPG processes because they do not exhibit natural mutual-exclusivity behavior and can potentially cause contention. You get the tool to perform this additional analysis by using the Atpg_analysis option with the set_drc_handling command. Buses in the pass category require no further checking.

Refer to "Bus Mutual Exclusivity Analysis" in the *Tessent Scan and ATPG User's Manual* for background information about the tool's bus mutual exclusivity checking. For more information about the measure_po statement, refer to the " Test Procedure File" in the *Tessent Shell User's Manual*.

Effect on Testability

Failure to satisfy this rule will result in the risk of bus contention during measurement of PO values (measure_po events in test procedures).

How to Debug E10 Violations

The occurrence message is:

```
Bus gate N (G) has possible contention on drivers G1 and G2. (E10-1)
```

N is the gate name of the bus gate, G is the gate ID number, and G1 and G2 are the gate ID numbers of the driver gates.

The summary message is:

```
There were N bus gates which may have possible contention (E10)
```

N indicates the number of bus gates failing the E10 rule.

If you see an "A" appended to the "E10" in the occurrence message (E10-1-**A** for example), it indicates the occurrence is due to the tool aborting the E10 check before it was completed. As these are "possible" but unproven E10s, the tool may be able to screen out some of them if you raise the abort limit. Refer to "Possible Resolutions" on page 2247 for more information.

How to Debug with DFTVisualizer

To view the location of an E10 DRC violation using DFTVisualizer, use the following command steps:

1. set_gate_level Primitive
2. open_visualizer

Then choose **Tools > Analyze DRC** from the DFTVisualizer menu, select the violation ID of the desired E10 DRC violation in the dialog box and click Analyze. The tool will automatically change the gate reporting to "parallel_pattern 0" and display the bus gate affected by that violation in the Debug window of DFTVisualizer. Think of "parallel_pattern 0" as an example of one possible pattern (created for DRC purposes only) that, if included in a test pattern set, would cause contention.

Tip: Alternatively, you can issue the analyze_drc_violation command with the *rule_id-occurrence#* argument at the tool's command line. For example, to display the first occurrence of an E10 violation:

analyze_drc_violation e10-1

Once the offending bus gate is displayed, trace backward from the bus gate. Figure 8-23 shows an example DFTVisualizer display of such a trace, where the starting point of the trace is the offending gate, /tsdo. Notice that TSDs /ix145 and /ix149 have conflicting values and both are enabled.

Figure 8-24. Example E10 Violation Trace in Debug Window of DFTVisualizer

You could use DFTVisualizer to trace back on the enable line of each TSD to the primary inputs and see that both tsden[1] and /scan_en are 1, causing the conflict.

How to Debug from the Tool Command Line

To view the location of the an occurrence of an E10 DRC violation from the tool command line, you can use the following command steps:

1. set_drc_handling E10 Warning
2. set_system_mode analysis
3. report_drc_rules E10-*occurrence#*
4. analyze_bus *gate_id*
5. set_gate_report Parallel_pattern *pattern_number*
6. report_gates *gate_id*

The following transcript excerpts show, for the design segment of Figure 8-23, an example of the use of this command sequence starting at step 3:

```
report_drc_rules e10-1
//  Warning: BUS gate /tsdo/dout[0] (314) has possible contention on
//           drivers 294 and 286. (E10-1)
analyze_bus 314
//  ATPG bus checking performed on /tsdo/dout[0] (314) for 2 bussed TSDs.
//  Failure occurred while checking TSDs 294 and 286 (data in
//  parallel_pattern 0).
set_gate_report parallel_pattern 0
report_gates 294 286
```

```
//  /tsdo/ix149 (294)  TSD
//      E      I  (1)  247-/ix310/Y
//      "D"    I  (1)  221-
//      Y      O  (X)  314-
//  /tsdo/ix145 (286)  TSD
//      E      I  (1)  161-/ix138/Y
//      "D"    I  (0)  213-
//      Y      O  (X)  314-
```

The conflicting values on the input ("D") of each TSD in this example while both are enabled (1 on the "E" input) is the source of the problem. Assuming the two TSDs should not be enabled simultaneously, a reasonable way to proceed is to choose one enable input and trace back through the gates driving it until you find the source of the 1. For example, you might first look at where the /tsdo/ix149 (294) input comes from:

```
report_gates -endpoints -backward 294
//  ----------------------------------------------------------------------
//  Begin backward trace for gate /tsdo/ix149 (294).
//  ----------------------------------------------------------------------
//  /tsden[1] (29)  PI
//      tsden[1]  O  (1)  160-/ix292/A
//  /scan_en (32)  PI
//      scan_en  O  (1)  163-/ix140/A0  161-/ix138/A1  162-/senmux/ix9/A1
//  /e4 (34)  PI
//      e4     O  (0)  163-/ix140/A1
//  /dataot/reg_q_0_ (373)  DFF
//      "S"    I  (X)  43-
//      R      I  (X)  157-
//      CLK    I  (X)  313-/dataot/ix104/Y
//      "D0"   I  (X)  255-
//      "OUT"  O  (1 [X])  127-  128-
//      MASTER  cell_id=8  chain=chain3  group=grp1  invert_data=TFFT
//  Number gates in trace = 4.
```

This shows that /tsden[1] is a 1. A similar backward trace from instance 286 to /tsden[0] shows that it too is a 1. If you could get a 0 on one of the /tsden inputs, it would remove the contention.

Possible Resolutions

Be aware you may not need to care about E10 violations in certain cases. If you are not concerned about contention after the capture clock is pulsed and you are satisfied with test coverage, you do not need to do anything. On the other hand, if the tool is rejecting many patterns due to contention during simulation and as a result, run time is excessive and coverage low, you may want to do something. By default, the tool will check for contention prior to the capture clock pulse in the capture cycle. If you are concerned about contention after the capture clock is pulsed, you need to direct the tool to check for this type of contention. The key point to remember is you only need to do something if you are not happy with what the tool does by default or the run time or coverage is unacceptable, in which case the following commands are helpful:

Note

These command suggestions will not make E10 rule violations go away, but will help you generate contention free patterns when E10 violations are present.

- add_input_constraints — Adds a pin constraint to a primary input pin. This command is useful if you determine from your troubleshooting and knowledge of the design that forcing a particular primary input pin to a specific value would resolve a contention issue. Typically, the need for such a constraint would have been anticipated during the design phase and perhaps simply overlooked when setting up for ATPG.
- add_atpg_functions and add_atpg_constraints—These commands allow you to create user-defined ATPG function constraints. This can be useful if you know where in the design constraints can be applied to remove contention. Constraints you apply will save time the tool would have to spend learning those locations through its own analysis.

  Following is an example of the use of these commands to specify constraints:

  ```
  add_atpg_functions func1 select1 tsden[0] ~tsden[1]
  add_atpg_constraints 1 func1
  ```

- set_drc_handling E10 Atpg_analysis — Directs the tool to perform additional analysis toward the end of DRC to determine if buses that the initial E10 DRC placed in the bidi category can cause contention. Increases the tool's DRC effort.
- set_drc_handling E10 -Mode Sequential — This command considers the inputs to a single level of sequential cells behaving as "staging" latches in the enable lines of tri-state drivers. All of the latches found in a back trace must share the same clock. There must also be only a single clocked data port on each cell, and both set and reset inputs must be tied (not pin constrained) to the inactive state. This check ensures that there is no connectivity from the cells in the input cone of the sequential cells and enable of the tri-state devices except through the sequential cells. Using this option will increase run time.
- set_abort_limit— Changes the abort limit. Increasing the abort limit can help screen out false E10 violations by enabling the tool to perform more extensive contention checking. This works for E10-*occurrence#*-A violations only.

Related Commands

add_atpg_constraints
add_atpg_functions
add_input_constraints
analyze_drc_violation
set_abort_limit
set_contention_check
set_drc_handling

E11

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

This rule checks for the ability of a bus gate to attain a Z state. This check analyzes each dominant strong bus to determine if conditions can place a Z value on the bus gate. As a result, the analysis places each bus in one of the following categories:

- **Pass** - Test generation analysis determines that no condition could place a Z value on the bus gate.
- **Fail** - Test generation analysis determines that conditions could place a Z value on the bus gate.
- **Abort** - Test generation analysis terminated while attempting to determine if conditions could place a Z value on the bus gate.
- **Bidi** - The bus is a bidirectional bus.

Buses in both the fail and abort categories violate this rule.

The default settings for this rule are ignore, noverbose, and atpg_analysis. You can change the handling with the set_drc_handling command. For more information on ATPG analysis, refer to "Turning on ATPG Analysis" on page 2136.

The occurrence message is:

```
BUS gate N (G) is capable of attaining a Z state. (E11-1-A)
```

N is the gate name of the bus gate, and G is its gate ID number. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N BUS gates capable of attaining a Z state. (E11)
```

N indicates the number of buses failing the E11 rule.

E12

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

This rule determines if the test procedures violate any ATPG constraints. The default handling for this rule is ignore. You can change the handling with the set_drc_handling command. For more information on ATPG analysis, refer to "Turning on ATPG Analysis" on page 2136.

The occurrence message is:

```
ATPG constraint violation on N (G) occurred at time T of P procedure.
(E12-1-A)
```

N is the gate name, G is its gate ID number, T is the simulated time period, and P is the procedure name. The -A following the violation ID number indicates the check aborted.

The summary message is:

```
There were N occurences of ATPG constraint violations in test procedures.
(E12)
```

N indicates the number of E12 rule violations.

E13

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Supported

Description

This rule determines if it is possible to satisfy both ATPG constraints and bus contention prevention (for buses that fail rule E10). The default settings for this rule are ignore and noverbose. You can change the handling with the set_drc_handling command.

The occurrence message is:

```
Contention prevention/ATPG constraints satisfiability check failed. (E13-
1)
```

This rule does not issue a summary message.

E14

Category: Extra

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

The expected simulation values listed in the test_setup procedure must match the actual simulated values.

E14 ensures the test_setup procedure is correct.

This DRC simulates the test_setup procedure when setup mode is exited and compares the simulated values with the expected values listed in the test_setup procedure.

Exception

Not applicable.

Messaging

If the test_setup procedure file fails E14, the following message displays:

```
Expect failed on <signal name> at time <time>. Expected <logic value>,
simulated <logic value>.  (E14-#)
```

This rule does not issue a summary message.

Troubleshooting

1. Use DFTVisualizer to compare the DRC simulated values from the test_setup procedure with the simulated values for associated nodes in the design.
2. Correct the expect statements in the test_setup procedure to match the DRC simulation values.

EDT Finder Rules (F Rules)

F rules verify the EDT logic connectivity and operation prior to scan chain tracing.

F3

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

The design must contain at least one state element able to change states during shift. F3 verifies that at least one state element in the design is clocked and not constrained to a fixed value.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
At least one state element should be able to change its state during
shift. None were found. (F3)
```

F4

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

EDT requires a design containing at least one decompressor. F4 verifies there is a decompressor in the design.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
No decompressors found. (F4)
```

F5

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

All defined channel inputs must drive a decompressor. F5 verifies that all defined input channels are driving an identified decompressor by tracing the path from the channel input to the decompressor. Figure 8-25 shows an error condition.

Figure 8-25. Blocked Decompressor Input

The error message is:

```
Channel input P (N) [of EDT block E] is not driving a decompressor.
Either no structural connection exists, or the path is blocked.
Trace stopped at state element(s) S (G). (F5-1)
```

Where P is the channel index and N the pin name, E is the name of an EDT block, S is an instance name and G is the corresponding gate ID number. Multiple state elements may be listed in case of fan-outs.

Troubleshooting

Check the procedure file for incorrect or missing shift procedure settings.

Related Topics

Test Procedure File

F6

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

Description

Channel input pins that control the decompressors must be defined. F6 traces each decompressor input pipeline to verify that decompressors are controlled by channel input pins, and verifies that the state elements in the channel input pipelines are properly clocked during shift. Figure 8-26 shows an error condition.

Figure 8-26. Blocked Channel Input to Decompressor

This check is performed even when F5 violations exist.

If the decompressor is driven by one or more defined channel inputs, each blocked input pipeline is reported as follows:

```
The input pipeline of decompressor D [of EDT block E] is broken between
decompressor state element L (G1) and the channel input after tracing N
state elements. The last state element reached is: M (G2). (F6-1)
```

If all input pipelines of a decompressor are blocked the following message is reported once per decompressor.

```
All input pipelines of decompressor D are blocked. One of the input
pipelines is broken between decompressor state element L (G1) and the
channel input after tracing N state elements. The last state element
reached is: M (G2). (F6-1)
```

If the clock input of a channel input pipeline is not set active during shift, the following message is reported:

```
The input pipeline of decompressor D [of EDT block E] is broken between
decompressor state element L (G1) and the channel input after tracing N
state elements. The clock inputs of state element M are never set active
during the shift procedure (F6-1)
```

Where D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G1 is the corresponding gate ID number, N is the number of found pipeline stages, M is the instance name of the last traced state element and G2 is its gate ID number.

F7

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Decompressors must be controlled by defined channel inputs. F7 traces all inputs to the decompressor to verify that they are connected to channel input pins.

If the decompressor is driven by at least one defined channel input, each rule violation is reported as follows:

```
The pin P is not a channel input but drives decompressor state element L
(G) of decompressor D [of EDT block E]. (F7-1)
```

If the decompressor is only driven by pins not defined as channel inputs, the following error message is reported once per decompressor:

```
None of the primary input pins driving decompressor D is defined as a
channel input. One of the traced pins is P. (F7-1)
```

Where P is a pin name, D is the internal identification number of the decompressor, E is the name of an EDT block, L is the instance name of the decompressor state element and G is the corresponding gate ID number.

Note

In the EDT mapping flow, the reported pins are channel inputs mapped to the top level of the design.

F8

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that all channel outputs are driven by a compactor. F8 traces all channel outputs to verify that they are driven by a compactor, and verifies that the state elements in the channel output pipelines are properly clocked during shift. Figure 8-27 shows an error condition.

Figure 8-27. Output Channel and Compactor Connection Blocked

The error message is:

```
The channel output P (N) [of EDT block E] is not driven by a compactor.
Either no structural connection exists, or the path is blocked. The last
state element reached while tracing back is: M (G). (F8-1)
```

If the clock input of a channel output pipeline is not set active during shift, the following message is reported:

```
Channel output P (N) [of EDT block E] is not driven by a compactor. Either
no structural connection exists, or the path is blocked. The clock inputs
of state element M (G) are never set active during the shift procedure
(F8-1)
```

If a primary input was reached the message is:

```
The primary input pin PI was reached while tracing back from channel
output P (N) [of EDT block E]. (F8-1)
```

If an unclocked element is driving the channel output, the message is:

```
N state elements drive the channel output but could not be identified as
mask hold registers (because they are not driven by mask shift registers)
nor as part of scan chains (because the clock inputs are never set active
during the shift procedure). One of those is:
M1 (G1).
```

```
...
N is the number of unclocked state element driving the channel output, M1
is the instance name of one of those, G1 is its gate ID number.
```

If the tracing problem is caused by a defined value on a channel output, the message is:

```
The channel output P (N) [of EDT block E] cannot be traced back because it
is constrained to V. An unconstrained simulation value of X is expected.
(F8-1)
```

If the tracing problem is caused by wrong values in mask hold registers, the message is:

```
Wrong values detected in mask hold registers after disabling EDT X masking
logic and simulating load_unload until first clock event in apply shift.
One of the failing state elements is: M (G). Simulated value: V1. Expected
value: V2.(F8-1)
```

Where P is the channel index and N is the pin name, E is the name of the EDT block, M is the instance name of the last traced state element, G is its gate ID number, PI is the name of the reached primary input pin, and V, V1, V2 are gate values.

F9

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that all compressed scan chains are connected to a decompressor. F9 traces all channel outputs to verify they connect to a decompressor. Figure 8-28 shows an error condition.

Figure 8-28. Blocked Scan Chain Path

The error message is:

```
Compressed scan chain driving channel output P (N) [of EDT block E]
blocked after tracing C cells. The last gate reached while
tracing back is: M (G).  (F9-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, C is the number of scan cells traced in the scan chain, M is the instance name of the last traced gate and G is its gate ID number.

F10

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the EDT logic found matches the EDT logic expected by the tool.

This DRC also verifies that the status of the low-power hardware is consistent with the user-specification of the set_edt_power_controller command in the dofile. If the tool identifies low-power hardware but the set_edt_power_controller command in the dofile disables it, the tool issues an error. If the tool is unable to identify any low-power hardware but the set_edt_power_controller command in the dofile enables it, the tool also issues an error.

Examples of error messages are:

```
The size I1 of decompressor D [of EDT block E] does not match the expected
size I2. (F10-1)

Low power hardware was found but should be disabled according to the
dofile. Verify that the hardware is disabled in the test procedures.
(F10-1)

The input taps of decompressor D [of EDT block E] do not match the ones
expected by the tool. For details about the found decompressor use
'report_edt_finder -Decompressors -Id D -Verbose'. (F10-2)

The expected decompressor taps of scan chain S [of EDT block E] do not
match with the taps of any found scan chain. (F10-3)

The input taps of channel input N (P) do not match the ones expected by the
tool. Verify that the channel inputs are defined in the correct order.
(F10-4)
```

Where D is the name of the decompressor, E is the name of an EDT block, S is the name of the scan chain, N is the index number of the channel input, and P is the pin name.

F11

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Note

report_drc_rules: Supported

Description

Verifies that all decompressors connect to at least one scan chain and that the path from the decompressor to the scan chains is not blocked due to wrong or missing shift procedure settings.

No correction is required if the decompressor is not needed for testing.

The occurrence message is:

```
Decompressor D has been found but is not used in the current mode. It
contains S state elements, one of which is N (G). (F11-1)
```

Where D is the internal identification number of the decompressor and S is the number of its state elements, N is the instance name of a decompressor state element and G is the corresponding gate ID number.

F12

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

Description

Verifies that all uncompressed scan chains are defined with the add_scan_chains command.

The warning message is:

```
An uncompressed scan chain has been found that was not defined with the
"add scan chain" command. Its scan input is P1, scan output is P2, and it
contains S state elements. (F12-1)
```

Where P1 is a primary input pin, P2 is a primary output pin and S is the number of state elements in the scan chain.

F13

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the decompressor, Xpress mask shift register state elements, and low-power shift registers are reset correctly prior to shift.

The error message in the case of the decompressor is:

```
The decompressor [of EDT block E] is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing decompressor state elements is: M (G). (F13-1)
```

In the case of an Xpress compactor, the state elements of the mask shift registers are checked in the same way. The error message is:

```
The mask shift register driven by channel input P (N) [of EDT block E] is
not reset correctly prior to shift. One of the failing mask shift register
state elements is: M (G). (F13-1)
```

In the case of a low-power shift register, the error message is:

```
The low power shift register driven by channel input P (N) [of EDT block
E]is not reset correctly prior to shift.
Simulated value: V1. Expected value: V2.
One of the failing low power shift register state elements is: M (G).
(F13-1)
```

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number, and V1 and V2 are gate values.

F14

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the state elements of the mask shift registers and the low-power shift registers hold their value during capture.

Only one violation per EDT block is reported.

The error message for mask shift registers (Xpress compactor) is:

```
The mask shift register driven by channel input P (N) [of EDT block E]
does not hold its value during capture. One of the failing mask/power
shift register state elements is: M (G). (F14-1)
```

The error message for mask shift registers (basic compactor) is:

```
The mask shift register [of EDT block E] does not hold its value during
capture. One of the failing mask shift register state elements is: M (G).
(F14-1)
```

The error message for a low-power shift register is:

```
The low power shift register driven by channel input P (N)
[of EDT block E] does not hold its value during capture. One of the
failing low power shift register state elements is: M (G). (F14-1)
```

In these messages, P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element and G its gate ID number.

F15

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that input pipeline state elements are initialized correctly. F15 verifies that any state elements that exist between a channel input and the EDT logic are initialized to the right value so that 0s get propagated into the EDT logic during the first shift.

Only one violation per EDT block is reported.

You cannot change the handling with the set_drc_handling command.

The error message is:

```
State elements in the input pipeline driven by channel input P (N) [of EDT
block E] are initialized to wrong values after test_setup and load_unload.
One of the failing input pipeline state elements is: M (G). Simulated
value: V1. Expected value: V2. (F15-1)
```

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, G is the gate ID number, and V1 and V2 are gate values.

F16

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Input pipeline state elements must preserve their value during capture and load_unload. F16 verifies that any state elements between a channel input and the EDT logic preserve the value shifted into them during capture and load_unload procedure. For additional information on pipelining, see "Ensuring Input Channel Pipelines Hold Their Value During Capture" in the *Tessent TestKompress User's Manual*.

Only one violation per EDT block is reported.

The error message is:

```
State elements in the input pipeline driven by channel input P (N) [of EDT
block E] do not preserve their value during capture and load_unload. One
of the failing state elements is: M (G). (F16-1)
```

Where P is the channel index and N the pin name, E is the name of the EDT block, M is the name of the failing state element, and G is the gate ID number.

F17

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the state elements of the mask hold registers and the low-power hold registers are clocked during load_unload.

Only one violation per EDT block is reported.

The error message reported when mask hold registers violate this rule is:

```
Mask hold register state elements [of EDT block E] are not clocked during
load_unload. One of the failing mask hold register state elements is:
M (G). (F17-1)
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] are not clocked
during load_unload. One of the failing low power hold register state
elements is: M (G).
(F17-1)
```

In these messages, E is the name of the EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F18

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the state elements of the mask hold registers load their values from the state elements of the mask shift registers before shift in the load_unload procedure; verifies that the state elements of the low-power hold registers load values from the low-power shift registers before shift in the load_unload procedure

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

```
Mask hold register state elements [of EDT block E] do not load values from
mask shift registers before shift in the load_unload procedure.
One of the failing mask hold register state elements is: M1 (G1).
The driving mask shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2.
```

The error message reported when a low-power hold register violates this rule is:

```
Low power hold register state elements [of EDT block E] do not load values
from low power shift registers before shift in the load_unload procedure.
One of the failing low power hold register state elements is: M1 (G1).
The driving low power shift register state element is: M2 (G2).
Simulated value: V1. Expected value: V2. (F18-1)
```

In these messages, E is the name of the EDT block, M1 is the name of the failing state element, G1 is its gate ID number, M2 is the name of the driving state element of the low-power hold register and G2 its gate ID number, and V1 and V2 are gate values.

F19

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the state elements of the mask hold registers and low-power hold registers preserve their values during shift.

Only one violation per EDT block is reported.

The error message reported for mask hold registers is:

```
Mask hold register state elements [of EDT block E] do not preserve their
values during shift. One of the failing mask hold register state elements
is: M (G).(F19-1)
```

The error message reported for low-power hold registers is:

```
Low power hold register state elements do not preserve their values during
shift. One of the failing low power hold register state elements is: M
(G). (F19-1)
```

In these messages, E is the name of an EDT block, M is the name of the failing state element of the low-power hold register and G is its gate ID number.

F20

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

The F20 design rule check verifies the masking modes of a compactor.

F20 programs the compactor to the various masking modes by directly loading the mask hold registers and tracing backward from the channel outputs (or from the first pipeline stage in case of a pipelined channel output) to verify that only the expected scan chains are driving the channel outputs.

If the F20 DRC has verified the masking modes (that is, if the check is not set to "ignore"), the tool skips the corresponding K22 DRCs.

Mask-all Masking Mode

If the compactor supports the mask-all masking mode, F20 verifies that all scan chains are masked in that mode; that is, the channel output receives a “0” (or a “1” in the case of an inversion). In the case of a violation, the error message is:

```
Mask-all masking mode: The channel output P (N) [of EDT block E] receives
an X although all scan chains driving it are expected to be masked. The
last reached gate while tracing back is: M (G) [which is the scan output
of scan chain 'S']. (F20-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, M is the instance name of the last traced gate and G is its gate ID numbe, and S is the name of the found scan chain. Note, the string “[which is the scan output of scan chain ‘S’]” is only reported if M is the scan output of a scan chain.

1-Hot Masking Mode

F20 verifies that only the expected scan chain drives the expected channel output. All other scan chains are masked. If the expected channel output is not driven by the expected scan chain the error message is:

```
1-hot masking mode for scan chain 'S1':
The channel output P (N) [of EDT block E] is not driven by gate M1 (G1)
which is the scan output of scan chain 'S1'.
The last reached gate while tracing back is:
M2 (G2) [which is the scan output of scan chain 'S2']. (F20-1)
```

Where P is the channel index and N is the pin name, E is the name of an EDT block, S1 is the name of the expected scan chain, M1 is the instance name of the scan output gate of scan chain

S1 and G1 is its gate ID number, M2 is the instance name of the last traced gate and G2 is its gate ID number, and S2 is the name of the found scan chain. Note, the string “[which is the scan output of scan chain ‘S2’]” is only reported if M is the scan output of a scan chain.

Xpress Masking Mode

If the compactor supports the Xpress masking modes, F20 verifies that only the expected scan chains drive the expected channel outputs. All other scan chains are masked. If the expected channel output is not driven by the expected scan chains, the error message is:

```
XOR masking mode K: The channel output P (N) [of EDT block E] is not driven
by the following scan chains:
S1, S2, …, SN as expected.
The last reached gate while tracing back is:
M (G) [which is the scan output of scan chain 'S']. (F20-1)
```

Where K is the index of the masking mode, P is the channel index and N is the pin name, E is the name of an EDT block, S1, S2, …, SN are the names of the expected scan chains, M is the instance name of the last traced gate and G is its gate ID number, and S is the name of the found scan chain. Note, the string [which is the scan output of scan chain ‘S’]” is only reported if M is the scan output of a scan chain.

F21

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Error

report_drc_rules: Supported

Description

The F21 design rule check verifies that all scan chains driving a compactor have the expected latency; that is, the expected number of pipeline stages between the scan output and the channel output.

In the case of a violation, the error message is:

```
Wrong number of pipeline stages between the scan chain 'S'
and the channel output P (N) [of EDT block E]: identified=N1 expected=N2.
The scan output of scan chain 'S' is M (G)
The following pipeline stages have been identified:
P1 (G1)
P2 (G2)
…
PN (GN) (F21-1)
```

Where S is the name of the scan chain, P is the channel index and N is the pin name, E is the name of an EDT block, N1 and N2 are the number of identified and expected pipeline stages, M is the scan output of the scan chain and G is its gate ID number, and P1, P2, …, PN are the identified pipeline stages and G1, G2, …, GN their gate ID numbers.

F22

Category: EDT

Contexts Supported: patterns -scan

Default Handling: Warning

report_drc_rules: Supported

Description

When two adjacent memory elements (source and sink) in the scan path are clocked by different shift clocks, the sink must not capture data from the source at the same time that the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan shifting when clock skew exists between the different shift clocks.

This DRC verifies this rule for EDT-related state elements. This DRC complements the T24 DRC that verifies this rule for the scan chains. Together, these two DRCs cover the complete scan path .

In the case of a violation, the error message is:

```
Two adjacent pipeline cells of channel input P (N)[of EDT block E]
are clocked on the same edge by different clocks.

N1 (G1), driven by clock N3 (G3), and
N2 (G2), driven by clock N4 (G4)

Consider adding a lockup cell between them, or modifying the clock timing.
(F22-1)
```

The first sentence of the error message depends on the location of the violation. Other possibilities are:

```
Two adjacent cells of the pipeline of channel input P (N) and the
decompressor [of EDT block E]…

Two adjacent cells of the decompressor [of EDT block E] and scan chain
'C1' …

Two adjacent cells of scan chain 'C1' and the compactor [of EDT block E] …

Two adjacent cells of scan chain 'C1' and the pipeline of channel output P
(N) [of EDT block E]…

Two adjacent cells of the compactor [of EDT block E]…

Two adjacent cells of the compactor and the pipeline of channel output P
(N) [of EDT block E]…

Two adjacent pipeline cells of channel output P (N) [of EDT block E]…
```

N1 and N2 are the instance names of the adjacent elements clocked by different clocks, N3 and N4 are the instance names of the clocks, G1-G4 are the corresponding gate IDs, P is the channel index and N the pin name, E is the name of an EDT block, and C1 is the name of a scan chain.

Note

If the F22 DRC has verified this rule (that is, the set_drc_handling command is not set to "ignore" F22 violations) the K21 DRC is skipped. This only applies to the pattern generation phase and not the IP creation phase.

How to Debug Using DFTVisualizer

To view the location of a F22 DRC violation using DFTVisualizer, use the following command:

analyze_drc_violation F22-1

This command invokes DFTVisualizer and opens both the Debug window and the Test Structures window. The Test Structures window shows a representation of the test structure and highlights the location of the two adjacent cells with a callout message of the cell's name as shown in Figure 8-29. The text pane displays the name of the cell, the type of cell, and, in the Analysis field, lists the controlling clock for that cell.

Figure 8-29. analyze_drc_violation F22 in Test Structures Window

In this example, the Test Structures window callouts indicate that the F22 violation occurs because a lockup cell is missing in the path from the decompressor to the scan chain. As shown in Figure 8-30, this typically happens if you use the skeleton flow and define incorrect clock timing for the first cell of the scan chain.

Figure 8-30. Callout Text for reg_lfsm_vec_lockup_3

The Debug Window shows the connected path between the two adjacent cells and displays in the callouts the two different clocks that pulse on the same edge. The gate report is set to the shift procedure so you can see that both cells capture on the same edge.

Figure 8-31. analyze_drc_violation F22 in Debug Window

This type of violation typically occurs because during IP creation the clock of the first scan cell in the chain is leading edge (010), but after synthesis the first cell is actually a trailing edge flop (101). When you are using the skeleton flow, it is important that the edge of the first and last cell is defined correctly or the tool may end up with an extra or missing lockup cell.

Solution

You can correct this violation by inserting a lockup cell between the adjacent memory elements reported in the violation message, or by regenerating the EDT IP using the correct edges for the first cell.

Flattening Rules (FN, FP, and FG Rules)

The flattening rule violations and their identification literals are divided into the following three groups:

- Net rules — FN1 through FN9.
- Pin rules — FP1 through FP13.
- Gate rules — FG1 through FG8.

You can use the set_flattener_rule_handling command to change the handling of the net, pin, and gate flattening rules.

FN1

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

By default, the tool ignores this rule in the RTL context. You can overwrite this default handling by using the set_flattener_rule_handling command.

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a module net is floating.

FN2

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a module net has a driver and a constant value property. The constant value property is ignored.

FN3

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

By default, the tool ignores this rule in the RTL context. You can overwrite this default handling by using the set_flattener_rule_handling command.

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an instance net is floating.

FN4

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an instance net is not used.

FN5

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule identifies a wired net with multiple drivers.

FN6

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a bus net attribute cannot be used.

FN7

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if two connected nets have inconsistent net attributes. Both inconsistent attributes are ignored.

FN8

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule identifies parallel wired behavior.

FN9

Category: Flattened Net

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the bus net has multiple bus keepers. The effects of the bus keepers are additive.

FP1

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the circuit has no primary inputs.

FP2

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the circuit has no primary outputs.

FP3

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a primary input drives logic gates and switch gates.

FP4

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a pin was moved.

FP5

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a pin was deleted by merging.

FP6

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule identifies merged wired in/out pins.

FP7

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule identifies merged wired input and output pins.

FP8

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if a module boundary pin has no name.

FP9

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an inout pin is used as output only.

FP10

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an output pin is used as an inout pin.

FP11

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an input pin is used as an inout pin.

FP12

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an output pin has no fan-out.

FP13

Category: Flattened Pin

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

By default, the tool ignores this rule in the RTL context. You can overwrite this default handling by using the set_flattener_rule_handling command.

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if an input pin is floating.

FG1

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the defining model of an instance does not exist. If you reduce the handling to warning or note, this primitive is treated as undefined.

FG2

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the feedback gate is not in feedback loop.

FG3

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the bus keeper has no functional impact.

FG4

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the RAM/ROM read attribute is not supported.

FG5

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the RAM attribute not supported.

FG6

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the RAM type is not supported.

FG7

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the netlist module has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

FG8

Category: Flattened Gate

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported. Use report_flattener_rules.

Description

This rule determines if the library model has a primitive that is not supported. If you reduce the handling to warning or note, this primitive is treated as undefined.

General Rules (G Rules)

G rules are general rules that identify inconsistencies in scan data and other definitions. G rules are the first rules checked during the DRC process and all violations generate error conditions that must be corrected. You cannot change the handling of these rules.

G1

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

Each defined scan chain group, except for “dummy,” must contain at least one scan chain. You can correct this error condition by either adding a scan chain to the group or by deleting the scan chain group. The error message is:

```
No scan chains have been defined for group N. (G1-1)
```

N is the name of the scan chain group, and G1-1 indicates the rule and violation ID numbers.

G2

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If you define scan chains and do not use the “dummy” scan chain option, you must define at least one clock. You can correct this error condition by either defining a clock that controls the defined scan chains or by deleting all scan chain groups. The error message is:

```
Scan chains exist but no clocks have been defined. (G2-1)
```

G3

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If the circuit has no memory elements, you cannot define clocks. You can correct this error condition by deleting all clocks. The error message is:

```
Clocks are defined but no memory elements exist in the circuit. (G3-1)
```

G4

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If the circuit has no memory elements, you cannot define scan chain groups. You can correct this error condition by deleting all scan chain groups. The error message is:

```
Scan groups are defined but no memory elements exist in the circuit.(G4-1)
```

G5

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If there are no RAMs in the circuit, you cannot define write control lines. You can correct this error condition by deleting all write control lines. The error message is:

```
Write controls are defined but no RAMs exist in the circuit. (G5-1)
```

G6

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If you define a linear feedback shift register (LFSR), you cannot use the “dummy” scan chain option. You can correct this error condition by either deleting all LFSRs or deleting the dummy scan chain group. The error message is:

```
Cannot use dummy scan chain with BIST LFSRs. (G6-1)
```

G7

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

The RAM/ROM instance name given on a preceding read_modelfile command must contain a single RAM or ROM gate. You can correct this error condition by using the correct RAM or ROM instance name for the read_modelfile command. Or, you can do nothing and re-invoke the rules checker, in which case, the tool will not use a modelfile for the intended RAM or ROM. The error message is:

```
Cannot use RAM/ROM modelfile M for invalid instance N. (G7-1)
```

M is the modelfile name, N is the instance name, and G7-1 indicates the rule and violation ID numbers.

G8

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

All ROM gates must have a defined initialization file, unless you use the random initialization option. You can correct this error condition by: specifying a modelfile in the model cell library, using the read_modelfile command to specify a modelfile, using random initialization, or changing the model cell library to treat the ROM gate as undefined. The error message is:

```
ROM initialization file not defined for N (G). (G8-1)
```

N is the instance name of the ROM, G is the gate ID number, and G8-1 indicates the rule and violation ID numbers.

G9

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

For all constrained scan cells identified by chain and position, the scan chain must be a valid scan chain, the position must be less than the length of the chain, and the scan cell must not be the same as another constrained scan cell. You can correct this error by identifying and correcting all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint position P for chain C. (G9-1)
```

P is the cell position number (0-based, where 0 is the scan cell closest to the scan-out pin), C is the scan chain name, and G9-1 indicates the rule and violation ID numbers.

G10

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

For all constrained scan cells identified by pin pathname, the pin must be a valid output pin of a cell, the pin must connect to a scan memory element through a path that only contains buffers and inverters, and the scan cell must not be the same as another constrained scan cell. To correct this error, you must identify and correct all invalid scan cell constraints.

The error message is:

```
Invalid cell constraint pin name P. (G10-1)
```

P is the pin pathname of an output pin of a cell, and G10-1 indicates the rule and violation ID numbers.

G11

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If you define a “dummy” scan chain group with a test procedure file, you cannot define any scan chains. The purpose of the dummy scan group is to provide the ability to use a **test_setup** procedure when no scan cells exist. To correct this error (if scan cells do exist), you should place the **test_setup** procedure in the test procedure file for a defined scan chain group.

The error message is:

```
Scan chains may not be defined when using dummy scan group procedure file.
(G11-1)
```

G12

Category: General

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

General rule 12 reports add_cone_blocks command pin-pathname violations because their checking is delayed until DRC is performed.

ICL Extraction Rules (I Rules)

Tessent Shell currently performs limited ICL Extraction rule checks on the design. Table 8-4 lists the ICL-specific DRC rules that check the mandatory pre-conditions the design must have in order to extract the ICL top level from the netlist.

Table 8-4. ICL Extraction DRCs

DRC	Description
I1	Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command. Also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module.
I2	Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port. Also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL.
I3	Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.
I4	This rule checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.
I5	Checks the consistency of specified pin constraints on ICL-attributed design instances.
I6	Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.

The ICL-specific DRC rules described in this section are based on the IEEE P1687 (IJTAG) proposed standard. These checks are only performed in the Tessent Shell patterns -ijtag context—see "ICL Extraction" in the *Tessent IJTAG User's Manual*.

Refer to "Design Rule Checks" in the *Tessent IJTAG User's Manual* for complete information.

I1

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Performs consistency checks on each design module that was mapped to an ICL module during the execution of the set_current_design command.

Specifically, the tool verifies that all ports on the mapped ICL module are present in the corresponding design module. If an ICL module port is not found in the corresponding design module, the tool issues the following error messages:

```
The port ‘P’ exists in the description of the ICL module ‘M1’ but not on
the matching design module ‘M2’.
```

Where:

‘P’ is the name of the ICL module port, ‘M1’ is the name of the ICL module, and ‘M2’ is the name of the design module matching the ICL module ‘M1’.

The tool also verifies that the direction of each port on the mapped ICL module is consistent with the direction of the corresponding port in the design module. If a design module port has a conflicting direction, the tool issues the following error messages:

```
DataInPort ‘PI’ exists in the description of the ICL module ‘M1’. However,
the corresponding port in the design module ‘M2’ is not of a
compatible direction. An ICL port of type DataInPort must map to a design
port that is either an input or bidirectional port.

DataOutPort ‘PO’ exists in the description of the ICL module ‘M1’.
However, the corresponding port in the design module ‘M2’ is not of a
compatible direction. An ICL port of type DataOutPort must map to a design
port that is either an output or bidirectional port.
```

Where:

‘PI’ and ‘PO’ are the names of the ICL module ports, ‘M1’ is the name of the ICL module, and ‘M2’ is the name of the design module matching the ICL module ‘M1’.

I2

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Verifies that a connection can be identified from an ICL-attributed pin to another ICL-attributed pin or to a top-level port.

This DRC violation typically occurs when a pin is unconnected. Specifically, this rule checks that every ICL-attributed instance input pin is connected to either an ICL-attributed instance output pin or a primary input port, and every ICL-attributed instance output pin is connected to either an ICL-attributed instance input pin or a primary output port.

The tool also checks if the simulated value for every ICL-attributed instance input pin is compatible with the tied value of 1 or 0 specified in the ICL using the allowed_tied_high or allowed_tied_low value for the connection_rule_option attribute, and reports an I2 violation if there is a mismatch. If the simulated value is X and the connection_rule_option is not allowed_no_source then the failure to drive the input pin is reported.

These checks are performed for all ICL input ports and for ICL output ports with the port functions DataOutPort and ScanOutPort.

If a connection cannot be identified, the tool issues the following error message:

```
The DataOutPort ‘Q’ on ICL-attributed design instance ‘I’, module ‘M’ does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
```

If the trace stopped at only one gate, which is always the case in the backward trace, the tool issues one of the following error messages which includes the gate information:

```
The DataOutPort ‘Q’ on ICL-attributed design instance ‘I’, module ‘M’ does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate ‘/I/Q’.

The DataOutPort ‘Q’ on ICL-attributed design instance ‘I’, module ‘M’ does
not trace to any primary or ICL module port.
Either no structural connection exists, or the path is blocked.
The forward trace stopped at gate ‘/I/Q’.
```

If there is a mismatch between the simulated value and the tied value, the tool issues one of the following error messages:

```
Mismatch between permissable tied value (1) declared by attribute
‘connection_rule_option’ for DataInPort ‘P’ on ICL-attributed design
instance ‘I’, module ‘M’ and the observed simulated value 0.
```

```
Mismatch between permissable tied value (0) declared by attribute
'connection_rule_option' for DataInPort 'P' on ICL-attributed design
instance 'I', module 'M' and the observed simulated value 1.
```

Where:

'P' is the name of the DataInPort, 'Q' is the name of the DataOutPort, 'I' is the instance name of the ICL-attributed start instance, 'M' is the module name of the ICL-attributed start instance. The DFTVisualizer can then highlight the starting instance 'I' and pin 'P' and the last gates that was reached during the tracing process.

I3

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Verifies that the port functions of an identified connection between an ICL-attributed port on an ICL-attributed module match the port function connection rules.

If the port functions do not match, the tool issues the following error message:

```
The connection between ScanOutPort 'SO' on ICL-attributed design instance
'I1', module 'M1' and ClockPort 'SI' on ICL-attributed design instance
‘I2', module 'M2' violates the ICL port function connection rules.

An instance ClockPort is only allowed to be driven by the following
sources: ClockPort on the parent module or ToClockPort reference from an
instance within the parent module. Where recognised values of the
function_modifier attribute are used to alter ICL port semantics they must
be applied consistently.
```

Where:

‘I1’ and ‘I2’ are the instance names of the ICL-attributed instances, ‘M1’ and ‘M2’ are the module names of the ICL-attributed instances, ‘SO’ is the name of the ScanOutPort on ICL-attributed design instance I1, ‘SI’ is the name of the ClockPort on ICL-attributed design instance ‘I2’.

The port functions of the top-level module ports are inferred from the identified connections according to Table 8-5 and are attached as port function attributes on the top level module ports.

Table 8-5. Port Connections

Start ICL-attributed Instance Port Function	Inferred Top Module Port Function
scan_in	scan_in
shift_en	shift_en
capture_en	capture_en
update_en	update_en
data_in	data_in
select	select
address	data_in
write_en	data_in
read_en	data_in

Table 8-5. Port Connections

Start ICL-attributed Instance Port Function	Inferred Top Module Port Function
scan_out	scan_out
data_out	data_out
to_shift_en	to_shift_en
to_capture_en	to_capture_en
to_update_en	to_update_en
to_select	to_select
to_ir_select	to_select
to_reset	to_reset
to_clock	to_clock

I4

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

This rule checks if an inferred port function and polarity for the primary input/output port is consistent for all found connections.

For example, a top-level module input port cannot be both a scan_in signal and a tck clock; it must be one or the other. If the inferred port function of port ‘SI’ is ‘scan_in’ due to an identified connection to port ‘P1’ and the inferred port function of port ‘SI’ is also ‘TCK’ due to another identified connection to port ‘P1’, the tool issues this error message.

```
Inconsistent port functions found for primary port /TCK.
Port function ScanInPort inferred due to a connection started at port 'SI'
on ICL-attributed design instance 'I1', module 'M1' is not consistent with
port function TCKPort inferred due to a connection started at port 'TCK'
on ICL-attributed design instance 'I2', module 'M2'.
```

Where:

‘SI’ is the ICL-attributed start port of the first connection, ‘I1’ is the instance name of the ICL-attributed start instance of the first connection, ‘M1’ is the module name of the ICL-attributed start instance of the first connection, ‘TCK’ is the inferred port function of the primary port due to the second connection, ‘I2’ is the instance name of the ICL-attributed start instance of the second connection, and ‘M2’ is the module name of the ICL-attributed start instance of the second connection.

If the polarity inferred due to the connection from the primary input ‘P’ to the module port ‘P1’ is different from the polarity inferred due to the connection from that primary input to another module port ‘P2’, the following error message will be issued:

```
Inconsistent polarity for reset port functions found for primary port P.
Polarity ‘Low’/’High’ inferred due to a connection started at port ‘P1’ on
ICL-attributed design instance ‘I1’, module ‘M1’ is not consistent with
the polarity ‘High’/’Low’ inferred due to a connection started at port
‘P2’ on ICL-attributed design instance ‘I2’, module ‘M2’.
```

Where:

‘P’ is the primary port name, ‘P1’ is the ICL-attributed start port of the first connection, ‘I1’ is the instance name of the ICL-attributed start design instance of the first connection, ‘M1’ is the module name of ‘I1’, ‘P2’ is the ICL-attributed start port of the second connection, ‘I2’ is the instance name of the ICL-attributed start design instance of the second connection, ‘M2’ is the module name of ‘I2’.

I5

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Checks the consistency of specified pin constraints on ICL-attributed design instances.

This DRC checks that the input ports specified in the forced_high_input_port_list are actually simulated to 1 in the stable_after_setup simulation conditions. It also checks that the input ports specified in the forced_low_input_port_list are actually simulated to 0 in the stable_after_setup simulation conditions. You typically need to adjust your test_setup procedure to correct this situation.

The error message is:

```
Input constraint verification failed for input pin ‘P’ on ICL-attributed
design instance ‘I’, module ‘M’.

The corresponding design module port was specified with the attribute ‘A’
in file ‘F’ on or near line ‘L’.

Expected V1, simulated V2.
```

Where:

‘P’ is the name of the failing pin, ‘I’ is the instance name of the ICL-attributed design instance of ‘P’, ‘M’ is the module name of ‘I’, ‘A’ is the input constraint attribute name (either forced_high_input_port_list or forced_low_input_port_list) defined for ‘P’ in the ICL file ‘F’ on line number ‘L’. ‘V1’ and ‘V2’ are the simulation values on the gate corresponding to the failing pin in the flat netlist.

I6

Category: ICL Extraction

Contexts supported: patterns -ijtag

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Checks all latches that have been previously identified during forward tracing of design instance pins with a scan out ICL attribute, and verifies that the backward tracing of the enable port of those latches reaches an identified primary port with TCK ICL attribute. In addition, all these latches have to be enabled by a low signal on the TCK port.

If no primary input port has been identified as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because no extracted
primary port with TCK attribute has been identified. (I6-1)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out functionality, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I).

If the enable port of a latch does not have a fixed simulation value and the backward tracing from the enable port does not end at the primary input port that has been identified before as TCK port, the tool issues the following error message:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because it requires an
inverted connection to the extracted primary port (P2) with TCK attribute.
Either no structural connection exists, or the path is blocked.
The backward trace stopped at gate (G). (I6-2)
```

Where:

(LI) is the instance name of a latch found during forward tracing of the pin with a scan out ICL attribute, (P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I), (P2) is the name of the primary input identified as TCK port during extraction and (G) is the name and ID of the last traced gate.

The DFT visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the last gates that were reached during the backward tracing process of the latch enable port.

If the enable port of a latch has a fixed simulated value, the following error message is issued:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because it requires an
inverted connection to the extracted primary port (P2) with TCK attribute.
The path is blocked by a signal setting. (I6-3)
```

Due to the blocking signal settings the DFT visualizer will highlight the gates that were reached during a structural backward tracing process of the latch enable port, the starting instance (LI), the identified primary port TCK (P2) and the scan out pin.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), and (M) is the module name of (I),

If the enable port of a latch does not have a fixed simulated value and the backward tracing ends at the primary input port that has been identified before as TCK port, but a low signal simulation does not enable the latch, the error message is:

```
Enable check of latch instance (LI) failed, driven by output pin (P) on
ICL-attributed design instance (I) of module (M) because a low signal at
the extracted primary pin (P2) with TCK attribute will not enable the
latch. (I6-4)
```

The DFT visualizer can then highlight the starting instance (LI), the identified primary port TCK (P2), the scan out pin (P) and the path from the enable port of the latch to the (P2) port.

Where:

(P) is the name of the start point of the forward tracing, (I) is the instance name of the ICL-attributed design instance of (P), (M) is the module name of (I),

ICL Semantic Rules (Semantic Rules)

The tool performs ICL semantic rule checks upon reading (using the read_icl command) an ICL file into the tool. These checks provide a more extensive check of the ICL upon reading the file or files into Tessent IJTAG. Basically, the ICL must conform to all of the semantic rules set out for the ICL in the IEEE P1687 (IJTAG) proposed standard. The semantic rule checks are as follows:

- **Parser** — When you read in an ICL module, the tool parses the file with a series of checks to detect for errors. The tool terminates the ICL parsing checks if any error is detected.
- **Module** — Once the complete module is available, the tool performs an extensive set of semantic checks using the totality of information contained exclusively within the module.
- **Hierarchical Module** — When the set_current_design command is issued the entire ICL module instance hierarchy is traversed and each module is rechecked, taking note of any instance parameter overrides on the module instances. Also, at this time those semantic rules that check the relationships between modules, such as module instance port binding semantics, are validated. If any of these checks fail then the ICL hierarchy is not well formed and can not be used, the set_current_design command will fail.
- **ICL Primitive Model** — A further set of checks are performed, using the more detailed ICL primitive model, these are performed at the primitive model bit level and generally check interconnection semantics locally, within a single module, or on the flattened ICL primitive representation, at the global level. Again, if any of these checks fail then the ICL hierarchy is not well formed and can not be used, the set_current_design command will fail.

Note

Many of the ICL Semantic Rules have multiple messages associated with them as some rules have broad classes of failure modes

ICL Semantic Rule 1

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source files have no parser syntax errors.

A failure of this semantic rule occurs if the input ICL source is not in compliance with the formal grammar for ICL. The accompanying message will show where the tool was when it encountered the incorrect syntax:

```
// Error: Incorrect character sequence. Encountered on or near line 5
// of file source.icl:
//                }
//                ^
//         ICL semantic rule 1
```

Note that the initial location of the incorrect syntax may actually be found prior to the location reported.

ICL Semantic Rule 2

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has no un-terminated multi-line comments.

ICL Semantic Rule 3

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module contains non-overlapping names within the single naming region used for the following objects:

- Ports
- Registers
- One-hot groups
- Multiplexers
- Alias description

ICL Semantic Rule 4

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an ICL module does not contain a duplicate attribute name for a single attributed declaration scope, such as a module or a register.

ICL Semantic Rule 5

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an ICL module has no duplicate RefEnum usage in those contexts where a binding to an enumeration can be made. ICL allows only a single enumeration set to be referenced.

ICL Semantic Rule 6

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file does not contain incorrectly cased ICL keywords. ICL keywords are case sensitive.

ICL uses mixed case keywords, that must be typed exactly as specified. This check will report a failure when the reading of an ICL source file encounters a context that is expecting one of these keywords but sees an incorrectly capitalized version of a valid keyword.

ICL Semantic Rule 7

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an indentifier does not exceed the maximum character length.

The maximum length for an identifier is 1024 characters.

ICL Semantic Rule 8

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has at least one module.

ICL Semantic Rule 9

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has no duplicate ICL module names.

All of the modules within a single ICL source file must be unique. A source file is allowed to provide a new definition of a module that has previously been read from another ICL source file; in this case the previous definition of the module is replaced with the new module definition, if the ICL source file is read without error.

ICL Semantic Rule 10

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an ICL module has no duplicate ICL parameter names within the definition of a single module.

ICL Semantic Rule 11

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an ICL module has no duplicate port group names within the definition of a single module.

ICL Semantic Rule 12

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has no duplicate entity element specifications.

For example, a DataOutPort declaration may provide a specification of the source value for the port using the Source keyword. This specification may only be made once within the declaration of the port.

ICL Semantic Rule 13

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the ICL Clock or ToClock declaration has an ActivePolarity specification value that is 0 or 1.

ICL Semantic Rule 14

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has no duplicate instance names within a module.

ICL Semantic Rule 15

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the ICL source file contains no directly self-referential instances.

A module is not allowed to instance itself. Circular instance hierarchies are also not allowed, and are caught by another rule check.

ICL Semantic Rule 16

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that each module instance within a one hot data group declares the address value for that instance, using the AddressValue specification.

ICL Semantic Rule 17

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that within a module definition, contained in an ICL source file, each module instance does not have multiple overrides for the same instance parameter name.

ICL Semantic Rule 18

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the specification of an address value for a module instance only occurs within module instances that are defined within a one hot data group. Module instances that occur outside of a one hot data group, in the main body of a module, can not have address value declarations, using the AddressValue specification.

ICL Semantic Rule 19

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file does not contain any scan registers without a scan input source.

Each scan register declaration must define the source of the scan input, using the ScanInSource specification.

ICL Semantic Rule 20

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file contains only supported ICL Constructs.

Tessent IJTAG does not yet provide support for the full ICL syntax. This rule will fail when the ICL reader encounters a piece of ICL syntax that is not currently supported.

ICL Semantic Rule 21

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the data register declaration within a one hot data group provides a definition of the address value for that data register, using the AddressValue specification.

ICL Semantic Rule 22

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a data register declaration that contains no write data source, which implies that the register is read only, also does not provide a write enable source. A read only data register does not require and must not have a write enable control.

ICL Semantic Rule 23

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a data register with an address value definition does not also have a write enable definition. Where a data register is defined within a one hot data group and has an address value the address and explicit write enable control will be used to control write accesses to the data register. Such an addressed data register does not require and must not define a write enable source.

ICL Semantic Rule 24

Reserved for future use.

ICL Semantic Rule 25

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that an addressed data register does not have a write data source.

An addressed data register will implicitly obtain the write data source from the data inputs of the module.

ICL Semantic Rule 26

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no duplicate enumeration declarations. Within an ICL module each enumeration value set must be given a unique name. The same enumeration element name can be reused, possibly with different value bindings, across multiple enumeration value sets.

ICL Semantic Rule 27

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file does not contain an enumerations using the Enum specification with a duplicate Enum item.

ICL Semantic Rule 28

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file does not contain an un-terminated string constant.

ICL Semantic Rule 29

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the iApplyEndState value, defined within an alias declaration, does not contain unknown bits.

ICL Semantic Rule 30

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks for inverted string constants.

Within a constant value expression the inversion operator (~) can only be applied to numeric constant values.

ICL Semantic Rule 31

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the constant value concatenation used to specify the value of a parameter definition in a module is comprised entirely of constant value elements that are of compatible type, either all string values or all sized numeric values.

ICL Semantic Rule 32

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a parameter definition, within a module declaration, does not contain a concatenation of values with an unsized numeric constant value.

ICL Semantic Rule 33

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that there are no self referential parameter definitions.

A parameter definition must be made in terms of explicit immediate constant values or references to other parameter definitions. When the parameter definition hierarchy is resolved it can not contain cyclical parameter references.

ICL Semantic Rule 34

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the size preamble for a sized number is a non-zero unsigned decimal number, where the size value is a deferred constant value or a parameter reference whose value is not verified until ICL Semantic Rule 51 is activated.

ICL Semantic Rule 35

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that only module parameters (not local parameters) are overridden in a module instance.

ICL Semantic Rule 36

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that any parameter present in an instance matches the definition in the instantiated module so that the parameter can be substituted wherever it is found in the module. Additionally, the parameter reference included in a parameter override element must match both the name and type of a parameter definition within the module being instantiated.

ICL Semantic Rule 37

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition contains at least one port definition. A module without any ports is not valid within ICL.

ICL Semantic Rule 38

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies the following:

- A module_def must include no more than one resetPort_def.
- There can be at most one TMSPort per module.
- There can be at most one TCKPort per module.
- There can be at most one WriteEnPort per module
- There can be at most one ReadEnPort per module.

ICL Semantic Rule 39

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that modules with more than one set of scan-in port name or scan out port name must include one port group definition to define the port names that are part of a common scan port.

ICL Semantic Rule 40

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module has at most one definition of a port using a scalar only name. If multiple bits are required, then they must be vector bits.

ICL Semantic Rule 41

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the module contains a well-defined set of vector port definitions. Within a module the same base name can be used when defining different vector port elements. However, all of the vector port definitions in a module must use a consistent index direction (either ascending or descending) and there can be no overlapping vector bit ranges. It is valid for a collection of vector port definitions with a common base name to define a non contiguous range of vector bits, so that the resultant vector aggregation has unbound segments. It is also valid to mix input and output port types within the vector port aggregation. For example:

- Multiple input port names that are vector_id can share the same SCALAR_ID as long as the indexes form a contiguous range and every index range is either ascending or descending.
- Port definitions cannot overlap.
- Multiple output port names that are vector_id can share the same SCALAR_ID as long as the indexes form a contiguous range and every index range is either ascending or descending.

ICL Semantic Rule 42

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that elements referenced within a PortGroup are ports.

ICL Semantic Rule 43

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the module that declares a scan input port for a client interface also declares a shift enable control port or a TMS port for that client scan interface. A client scan interface is any scan interface that doesn't contain host scan interface control ports.

ICL Semantic Rule 44

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a scan-in port is present in the module if the module declares either a scan register or a scan out port.

ICL Semantic Rule 45

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a module with a ToIRSelectPort also has a TMSPort.

ICL Semantic Rule 46

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a module that is not a black box module (so it is a module that has declarations other than just ports) defines sources for the output ports that require sources to be specified. Ports of type ScanOutPort, ToResetPort and ToSelectPort require a source specification when used in modules that are not black box modules.

ICL Semantic Rule 47

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a module that is not a black box module (so it is a module that has declarations other than just ports) defines sources for the ToClockPorts that are singular or for the primary element of a differential clock pair.

ICL Semantic Rule 48

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies the input port bindings on an instance within a module. Each bit of each input port binding must correspond to an input port bit definition on the instantiated module. Where an input port binding has a vector range constraint the underlying module port must have a port definition with a compatible definition.

ICL Semantic Rule 49

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that all module input ports that require input port bindings on an instance of a module have the bindings specified. Table 8-6 lists the port types that require input binding.

Table 8-6. Port Types and Required Input Binding

Port Type
ShiftEnPort
AddressPort
WriteEnPort
ReadEnPort
ScanInPort[1]
ClockPort[2]

1. When singular or the primary part of a differential clock pair
2. When singular or the primary part of a differential clock pair

ICL Semantic Rule 50

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that each bit of the module instance input port bindings within a module is specified only once. It is not valid to specify more than one input source binding for any bit of a module input port in a module instance.

ICL Semantic Rule 51

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the width of a sized constant value is greater than 0 (zero).

This check is similar to ICL Semantic Rule 34, however here the rule also considers the actual resolved values of any parameter references.

ICL Semantic Rule 52

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the size of a constant value expression or a reference to a named element within the module is compatible with the size expected at the location that the expression or reference is used.

For example a data register source value must be no wider than the width of the data register itself. This check takes into account unsized expression concatenation filling and truncation behaviors when considering the size compatibility.

ICL Semantic Rule 53

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the name referenced from a parameter reference corresponds to the name of a parameter or a local parameter within the same module.

ICL Semantic Rule 54

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the type of a parameter being de-referenced matches the type expected by the context of the point of de-reference. For example, within a range constraint of a vector signal “foo[$p]” the type of the parameter p must be an unsigned integer.

ICL Semantic Rule 55

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the parameter reference included in a parameter override element matches both the name and type of a parameter definition within the module being instantiated.

ICL Semantic Rule 56

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that any instance output port reference matches an output port on an instance within the module.

ICL Semantic Rule 57

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that any reference to an element as an rvalue makes reference to a name that exists within the same module, or an output or a module instantiated within that module. It is not permitted to make reference to hierarchical aliases. Where the reference uses a vector range constraint, it must actually reference an object that is itself a vector entity with a compatible range. References to module local names must resolve correctly to a readable entity.

ICL Semantic Rule 58

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that for a known size target, only one unsized number is included with a group of sized numbers. For an unknown size target, no unsized number can be included in concatenation.

ICL Semantic Rule 59

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a differential inverse clock is only an inverse of a primary clock. Additionally, the clock port name, an input port connection of a clock port name is required and must be a concat_clock_signal with a matching width and, if differential, a corresponding differential source.

ICL Semantic Rule 60

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that all instances are instances of modules for which a module definition has been read. It is not valid to use an ICL module hierarchy that contains references to modules that have not been defined.

ICL Semantic Rule 61

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that attribute values are either strings or sized binary values.

ICL Semantic Rule 62

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that when the iApplyEndState option is present, each bit of the resolved alias is a suitable target for an iWrite, specifically a DataInPort or a register bit.

ICL Semantic Rule 63

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies for a resolved alias with iApplyEndState value that includes register bits that each such bit has a reset value, and that the reset value matches the iApplyEndState value.

ICL Semantic Rule 64

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that clock inversion matches clock polarity.

ICL Semantic Rule 65

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the source of an rvalue is of a suitable type to drive the location that the rvalue is being consumed.

ICL Semantic Rule 66

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that no component has itself as a member of its fanin (scan).

ICL Semantic Rule 67

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a ToIRSelectPort only connects to the select input of a 2:1 ScanMux.

ICL Semantic Rule 68

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that within each module_def that all scan signals have a path from at least one scanInPort_name and to at least one scanOutPort_name.

ICL Semantic Rule 70

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a scan interface is composed of the correct subset of signals along with control signals related by direction. Specifically:

- Scan Client Interface — {ScanInPort, ScanOutPort, SelectPort, CaptureEnPort, ShiftEnPort, UpdateEnPort}
- Scan Host Interface — {ScanInPort, ScanOutPort, ToSelectPort, ToCaptureEnPort, ToShiftEnPort, ToUpdateEnPort}
- TAP Client Interface — {ScanInPort, ScanOutPort, TMSPort, TRSTPort}
- TAP Host Interface — {ScanInPort, ScanOutPort, ToTMSPort, ToTRSTPort}

Scan client, scan host, TAP client and TAP host ports may not be mixed.

ICL Semantic Rule 73

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that within non-target modules, a scan client interface has the following:

- One scanOutPort_name
- Zero or one selectPort_name
- Zero or one shiftEnPort_name

Checks that within non-target modules, a scan host interface has the following:

- Zero or one scanOutPort_name
- Zero or more toSelectPort_name
- Zero or one shiftEnPort_name

ICL Semantic Rule 74

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a host scan interface has no more than one bit of type ToShiftEnPort and no more that one bit of type ScanOutPort.

ICL Semantic Rule 75

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the instance output referenced from the Port specification of a one hot scan group or one hot data group is a port with an enable control or that is a port that is driven by a one hot scan group or one hot data group, respectively.

The net effect of this check is to ensure that any port reference will be a reference to an element that has a tri-state driver, either directly (because the port is tri-state enabled) or indirectly because it references a port that is driven by a one hot group.

ICL Semantic Rule 76

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies the following:

- Data storage elements controlling scan mux must have ResetValue.
- Data storage elements enabling ScanOutPort must have ResetValue.

ICL Semantic Rule 77

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a scanMux or one hot scan group does not deselect a controlling ScanRegister.

ICL Semantic Rule 78

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a ScanRegister that drives either the data input of a ScanMux whose select input comes from a ToIRSelectPort must be in the active scan chain in either one or the other state of the ToIRSelect signal, but not both.

The two inputs of the ToIRSelect scan mux are considered up until the point at which they reconverge. Scan registers in one path can not be in the other path.

ICL Semantic Rule 79

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

When the iApplyEndState option is present, then multiple iApplyEndState values must not specify incompatible values for any target. If there are multiple iApplyEndState values for a single target bit then they must all be of the same value.

The ultimate source of the iApplyEndState value must come directly from a data register, scan register or primary input, there can be no other intervening logic in that path.

The tool creates nets when creating the ICL primitive model representation of your design. These names are formed from a prefix that is taken from the ICL or ICL primitive model source name, and a suffix +NN.

For example, the names generated inside the representation of LogicSignal fooBar are as follows:

fooBar+0 ... fooBar+10

and so on.

This allows you to read ICL primitive model error messages and tie things back to the ICL source.

ICL Semantic Rule 80

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that no component has itself as a member of its fan-in (scan).

ICL Semantic Rule 81

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

The eventual driving source of a port of type {ReadEnablePort, WriteEnablePort or AddressPort} can only be a primary input or a scan register.

ICL Semantic Rule 82

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

The value associated with a clock frequency adjustment, either multiplication or division, must be a positive integer (greater than 0).

ICL Semantic Rule 83

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

The names associated with a hierarchical alias definition must all be resolvable (they must exist) to entities of an appropriate type and refer to things that have definitions compatible to their usage within the hierarchical alias definition. For example if the reference in the alias definition includes a range constraint then the resolved entity must be a vector object with compatible range definition.

ICL Semantic Rule 84

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that there are no unknown bits in a constant value expression.

ICL Semantic Rule 85

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks if there is a LogicSignal expression containing an operand with unknown bits.

ICL Semantic Rule 86

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks if there is a LogicSignal expression containing operands with incompatible widths. For example,1'b0 | 2'b11.

ICL Semantic Rule 87

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks for problems with enumeration references. For example, an enum value is not defined or a register has no enum references but uses it.

ICL Semantic Rule 88

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that there are no circular instance hierarchies in the ICL design.

If a circular loop is found, the error message also indicates the location of the loop.

```
//  Error: Loop in module instantiation tree:
//            Module ‘m1’ has instance ‘i2’ of module ‘m2’.
//            Module ‘m3’ has instance ‘i1’ of module ‘m1’.
//            Module ‘m2’ has instance ‘i3’ of module ‘m3’.
//        ICL hierarchies with circular instance loops are not supported.
Found on or near line 12 of file ‘../data/icl/test.icl’.
//         ICL semantic rule 88
```

This is a module level hierarchical check that is performed only once the set_current_design command has been issued.

ICL Semantic Rule 89

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

ICL connectivity extraction can be directed by the value of certain special attributes associated with the ICL ports for which a connection is being extracted, such as the ‘connection_rule_option’ attribute; for more information about directing ICL extraction with ICL attributes see “*Attributes of the ICL Extraction Flow”* semantic in the *Tessent IJTAG User’s Manual*. This semantic rule checks that the ICL port connection rule attribute has a value that is compatible with the port direction and will fail if this is not the case.

An error message is issued if the attribute ‘connection_rule_option’ has a value of ‘allowed_no_destination’ for an input port, or one of the following values for an output port: ‘allowed_no_source’, ‘allowed_tied’, ‘allowed_tied_low’, ‘allowed_tied_high’.

For example:

```
//  Error: ICL module ‘block1’ declares the port ‘broken_i1’ with
attribute ‘connection_rule_option’ that has the invalid resolved value
‘allowed_no_destination’. For an input port the value must be one of:
‘allowed_no_source’, ‘allowed_tied’, ‘allowed_tied_high’ or
‘allowed_tied_low’.
//  Error: ICL module ‘block1’ declares the port ‘broken_o1’ with
attribute ‘connection_rule_option’ that has the invalid resolved value
‘allowed_no_source’. For an output port only the value
‘allowed_no_destination’ is permitted.
//  Error: ICL module ‘block1’ declares the port ‘broken_o2’ with
attribute ‘connection_rule_option’ that has the invalid resolved value
‘allowed_tied’. For an output port only the value ‘allowed_no_destination’
is permitted.
//  Error: ICL module ‘block1’ declares the port ‘broken_o3’ with
attribute ‘connection_rule_option’ that has the invalid resolved value
‘allowed_tied_low’. For an output port only the value
‘allowed_no_destination’ is permitted.
//  Error: ICL module ‘block1’ declares the port ‘broken_o4’ with
attribute ‘connection_rule_option’ that has the invalid resolved value
‘allowed_tied_high’. For an output port only the value
‘allowed_no_destination’ is permitted.
```

ICL Semantic Rule 90

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks if there is a problem with the tessent_scan_register attribute.

A scan register with the tessent_scan_register attribute is not allowed to specify a capture source, reset value or default load. If a scan register with the tessent_scan_register attribute specifies a range constraint, it must be in descending order with a right index of 0. A scan register with the tessent_scan_register attribute cannot be the driver of a data out port or a logic signal.

ICL Semantic Rule 91

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks if there is a problem with the tessent_signature attribute.

If the module contains a tessent_instrument_type attribute which defines the module to be an instrument of type Mentor, the tessent_signature attribute must be present and must contain a valid signature.

ICL Semantic Rule 92

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no duplicate access link name declarations. Within an ICL module each access link must be given a unique name.

ICL Semantic Rule 93

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no duplicate access link instruction declarations. Within an ICL module each access link instruction must be unique.

ICL Semantic Rule 94

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains only one specified BSDL entity in the access link. Within an ICL module a BSDL entity can be specified only once in an access link.

ICL Semantic Rule 95

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode scan path declarations. Within an ICL module an opcode scan path can be specified only once in an access link.

ICL Semantic Rule 96

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no duplicate access link opcode active signal set declarations. Within an ICL module an opcode active signal set can be specified only once in an access link.

ICL Semantic Rule 97

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains no access link declarations if the module does not specify the top module. Only the top ICL module can have access link declarations.

ICL Semantic Rule 98

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that a module definition, within an ICL source file, contains access link declarations that are associated with valid paths.

ICL Semantic Rule 99

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that mux selection conditions are not made redundant by previous selections in the same mux.

ICL Semantic Rule 100

Category: ICL Semantics

Rule Check Type: Parser

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Checks that the ICL source file has no duplicate chain names within a scan interface.

ICL Semantic Rule 101

Reserved for future use.

ICL Semantic Rule 102

Category: ICL Semantics

Rule Check Type: Module

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies the following:

- Modules containing readable addressable data registers must have a ReadEnPort.
- Modules containing writable addressable data registers must have a WriteEnPort.

ICL Semantic Rule 103

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a module with one or many pairs of ports with function ScanInPort and ScanOutPort defines as many ScanInterface statements as there are scan interfaces in the module.

ICL Semantic Rule 104

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Each ‘scan client’ ScanInterface statement in a blackbox module must include a valid DefaultLoadValue definition, which reflects the length of the scan chain behind each ScanInterface after a global reset.

ICL Semantic Rule 105

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that the ports in a ScanInterface are as follows:

A Scan Client Interface shall have

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function ShiftEnPort and/or one or many ports with function SelectPort
- zero or one port with function CaptureEnPort
- zero or one port with function UpdateEnPort

A Scan Host Interface shall have

- one port with function ScanInPort and/or one port with function ScanOutPort
- one port with function ToShiftEnPort and/or one or many ports with function ToSelectPort
- zero or one port with function ToCaptureEnPort
- zero or one port with function ToUpdateEnPort

A TAP Client Interface shall have

- one or many pairs of ports with function ScanInPort and ScanOutPort
- one port with function TMSPort
- (optional port with function TRSTPort)

A TAP Host Interface shall have

- one port with function ScanInPort and/or one port with function ScanOutPort
- one port with function ToTMSPort
- (optional port with function ToTRSTPort)

ICL Semantic Rule 106

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that a chain definition of an ICL module scan interface only contains ports of the type ScanInPort and ScanOutPorts

ICL Semantic Rule 107

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that only scan client and TAP client interfaces can have scan chain definitions. Scan host and TAP host interfaces cannot have scan chain definitions.

ICL Semantic Rule 108

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that if a client scan interface has more than one ScanInPort/ScanOutPort pair, then each pair shall be declared in a scan chain definition.

ICL Semantic Rule 109

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that if a scan chain definition is present, there are no ScanInPort, ScanOutPort or DefaultLoadValue statements outside the scan chain definition.

ICL Semantic Rule 110

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that for every scan chain definition of an ICL module's scan interface, a ScanInPort/ScanOutPort pair is defined. The ports of this chain cannot be used in any other scan chain definition in the same scan interface.

ICL Semantic Rule 111

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies the following:

- Data storage elements controlling SelectPorts of ScanInterfaces must have ResetValue.
- Data storage elements controlling ToSelectPorts of ScanInterfaces must have ResetValue.

ICL Semantic Rule 112

Category: ICL Semantics

Rule Check Type: ICL primitive model

Contexts supported: All contexts

Default Handling: Error (severity cannot be changed)

report_drc_rules: Not supported

Description

Verifies that no binary digits of value 1 are truncated from a sized constant number.

If the number of binary digits of a number's binary representation is larger than the number of binary digits defined by the size constant, then the leftmost digits are truncated until it matches the size constant. Truncating a digit of value 1 generates an error.

EDT Rules (K Rules)

EDT rules only apply to designs that incorporate EDT technology. They are generally performed by Tessent TestKompress when switching from Setup system mode. The tool also performs some checks at other times such as in response to a write_edt_files command or before writing EDT files. These rule checks find EDT design definition inconsistencies.

The following subsections describe each of the EDT rules and the special handling you can set for them.

K1

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

EDT requires a circuit with scan chains. This rule check verifies that the design contains scan chains. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K2

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

EDT does not support designs with more than one scan group. This rule check verifies that the design defines exactly one scan group. The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K3

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

This rule check verifies that the scan chain input and scan chain output pin of each scan chain are both connected to either top-level ports or to internal pins. The following are valid ways to connect the scan chains:

- In the IP Creation Phase, all of the scan chains must be connected to primary inputs (PIs) and primary outputs (POs).
- In the Pattern Generation Phase, the scan-in and scan out pin of each chain must be connected either to a PI and PO (for an uncompressed scan chain) or to internal nodes (for a compressed scan chain driven by and observed through the EDT logic).

The handling for this rule violation is error. You cannot change the handling with the set_drc_handling command.

K4

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

Description

The EDT decompressor provides test patterns through unidirectional outputs to the inputs of the internal scan chains. Similarly, the EDT compactor accepts test responses through unidirectional inputs from the scan chain outputs. This rule check verifies that bidirectional top-level scan chain pins do not exist. The default handling for this rule violation is error.

K5

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This rule check verifies the existence of all required EDT control and channel pins at the top level of the design. The default handling for this rule violation is error.

If K5 reports as nonexistent an EDT control or channel pin you believe exists, it is typically because the pin does not exist at the *top* level of the design from which the tool can control the pin, or because the pin's current name differs from the pin name the tool expected. This difference could occur, for example, if you manually edited the pin name within the netlist after the IP Creation Phase. Use the set_edt_pins command to provide the tool with the correct pin name in these cases.

Note

You can use the set_edt_pins {edt_pin} - option to explicitly tell the tool that an EDT signal is driven internally and does not have an associated top-level pin. Specifying this option prevents the tool from generating a K5 violation for that pin.

K6

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

EDT pins, when shared with a design's functional pins, must be shared with a functional pin of the same type. For example, when an EDT input pin is shared with a functional pin, the functional pin must be an input. Similarly, an EDT output pin must only be shared with a functional output pin. This rule check verifies that each EDT pin that is shared with a functional pin is of a type consistent with the functional pin. Sharing with bidirectional pins is not allowed, and is covered by other design rule checks.

The K6 rule check also verifies the EDT control and channel pins have valid names. To be consistent with Tessent FastScan and to provide immediate feedback if you change the default pin names, Tessent TestKompress checks the validity of the pin names and pin directions when you enter a set_edt_pins command. Because the default EDT pin names might also conflict with names of existing pins, the tool performs all checks again when you switch from setup mode to analysis mode. The default handling for this rule violation is error.

The tool performs K6 rule checks at three different times. The items checked at each time by this rule are as follows:

These six requirements are checked immediately after the tool reads an EDT pin name. An error message results if the stated requirement is violated:

1. The pin name must not be hierarchical.
2. If the pin name is shared with a core pin, the directions must match.
3. If the pin is part of a bus, the base name for the pin must not be the same as the name of a core pin.
4. The pin name must not be the same as the base name of a core bus.
5. If the pin name matches a core bus pin name, the index must be within the same range.
6. If a pin name is a bus bit, the bus must exist in the core. Presently, bus-based EDT pins are not supported.

In addition to the preceding six requirements, the K6 rule checks the following requirements during DRC:

1. Each EDT pin must not be shared with either another EDT pin or with a restricted core pin. For information about restricted pins, refer to "EDT Control and Channel Pins" in the *Tessent TestKompress User's Manual*.
2. Two EDT pins must not share the same name.
3. An EDT output channel pin must not be shared with a bidi.

Finally, when you write out the files that implement the EDT IP using the write_edt_files command, the K6 rule results in a warning message for either of the following conditions:

1. The EDT clock pin is shared with a core bus pin.
2. An EDT pin is shared with a core pin that is empty (no fan-ins or fanouts).

K7

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

Description

EDT does not support either bidirectional or tristate scan channel output pins. This is because Tessent TestKompress includes a multiplexer between the EDT IP and the output pad when a scan channel output pin is shared with a functional output pin. This rule check verifies that neither bidirectional nor tristate scan channel output pins exist. The default handling for this rule violation is error.

K8

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Description

This rule check verifies that neither bidirectional scan channel input pins nor bidirectional EDT control pins exist. This is necessary because the EDT hardware has no control over the signal direction of bidirectional pins, and they might be driven by the core logic during load_unload, resulting in contention problems. The default handling for this rule violation is warning.

K9

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Description

Top-level scan chain pins cannot be shared with a design's functional pins. This is because Tessent TestKompress connects the scan chain pins to the EDT IP and they are no longer available as primary inputs or outputs in the new top level. This rule check verifies that scan chain pins are not used as functional pins. The default handling for this rule violation is warning.

The rule check assumes pin sharing in the following situations:

- Scan inputs—If a scan input drives a primary output, drives more than one scan cell, or is a bidirectional pin, pin sharing is assumed.
- Scan outputs—If the last scan cell of a scan chain does not drive other scan cells, and also drives exactly one primary output, pin sharing is assumed.

Note

The K9 rule check will not catch all cases of inappropriate pin sharing (the following discussion describes why this is the case). Therefore, be sure during scan insertion that you insert dedicated scan pins. This is true even if the output of the last scan cell in a scan chain is connected directly to a primary output.

In some cases, a gate-level netlist does not contain the information to enable Tessent TestKompress to determine if there is pin sharing. For example, Figure 8-32 shows a design where the last cell of a scan chain drives functional logic as well as the scan output pin. It is impossible to extract the information from the gate-level netlist about whether the pin was there before scan insertion or has been introduced for scan purposes.

Figure 8-32. Failing identification

Figure 8-33 shows another case where the K9 rule check cannot determine if pin sharing occurs. The scan cell contains a logic gate that drives the scan output. By examining the netlist, it cannot be determined if the logic gate is part of the scan cell or part of functional logic. Therefore, the design rule check would not detect the sharing.

Figure 8-33. Common cell structure

For the same reason, the design rule check would fail also for the circuit shown in Figure 8-34.

Figure 8-34. Failing identification

K10

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Description

Internal scan chain pins should not be part of a functional bus. Scan chain pins that are part of a functional bus are not connected to the EDT wrapper during pattern-generation phase. A DRC violation is reported if top-level scan chain pins are part of a functional bus.

Note

Top-level scan chain pins can be part of a pure scan bus.

K11

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

Description

The EDT clock, update, and bypass pins must not be shared with any other clock or RAM control signal. The reasons for this restriction are as follows:

- If the EDT clock or EDT update are shared with a scan clock, the scan cells would be disturbed during the load_unload procedure.
- If the EDT clock, update or bypass are shared with RAM control signals, RAM sequential patterns and multiple load patterns may not be applicable.
- If the EDT clock or EDT bypass are shared with a non-scan clock, the test coverage might decrease because the EDT clock is constrained to its off state during the capture cycle.

This rule check verifies that the EDT clock, update, and bypass pins are not shared with any other clock or RAM control signal.

K12

Category: EDT

Contexts Supported: dft -edt

Default Handling: Warning

report_drc_rules: Supported

Description

To guarantee that the EDT IP is not disturbed during capture, the EDT clock must be constrained to its off-state during the capture cycle. This rule check verifies that the EDT clock pin does not drive any logic.

Note

If the EDT clock is shared with a functional pin, the pin constraint may result in lower test coverage.

K13

Category: EDT

Contexts Supported: dft -edt

Default Handling: Note

report_drc_rules: Supported

Description

This rule check reports all pins that will be added to the EDT top-level wrapper in order to implement the EDT hardware.

Note

The tool reports scan channel and control pins as new pins even if a scan chain pin with the same name exists in the core circuit; otherwise that scan chain pin would not have been connected to the wrapper.

K14

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

In order to prevent EDT from assigning values to the scan chain inputs during the capture cycle, all internal scan chain inputs are automatically constrained to X by EDT. This rule check verifies that there are no user-defined constraints on the scan chain inputs that are different from TIE-X.

K15

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

set_drc_handling: Not supported

Description

This DRC does the following:

1. Verifies that all scan channels have the proper top-level pins. All scan channels, except for blocks set up for input channel broadcasting or input channel sharing, require a dedicated top-level pin.
2. Verifies that all EDT control pin sharing is set up properly.
3. Verifies that the internal scan chains driven by the EDT decompressor are not sharing scan inputs.

The tool reports error messages slightly differently depending on whether the EDT block is inside a core. If an EDT block is not inside a core, the tool uses a format that only includes the block name as shown in the following example error messages.

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//        Block "blk1" EDT  clock
//        Block "blk1" EDT update
```

- Verifies that the top-level input port does not connect to different control signals (like clock and update) on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_clock": (K15-1)
//        Block "blk1" EDT  clock
//        Block "blk2" EDT update
```

- Verifies that the top-level input port does not connect to the control signal and channel input of the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//        Block "blk1" EDT  update
//        Block "blk1" EDT channel 1 input
```

- Verifies that the top-level input port does not connect to the control signal and channel input of different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_a": (K15-1)
//        Block "blk1" EDT  update
//        Block "blk2" EDT channel 1 input
```

- Verifies that the top-level input port does not connect to different channel inputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//        Block "blk1" EDT channel 1 input
//        Block "blk1" EDT channel 2 input
```

- Verifies that the top-level output port does not connect to different channel outputs on the same block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//        Block "blk1" EDT channel 1 output
//        Block "blk1" EDT channel 2 output
```

- Verifies that the top-level output port does not connect to channel outputs on different blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_out1": (K15-1)
//        Block "blk1" EDT channel 1 output
//        Block "blk2" EDT channel 1 output
```

- Verifies that a top-level input port does not broadcast to both compressed and uncompressed blocks. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "scan_in1" for EDT channel and
//        uncompressed scan chain: (K15-1)
//        Block "blk1" EDT channel 1 input
//        Scan chain "chain1" input
```

- Verifies that two scan chains driven by the EDT decompressor cannot share the same scan input pin. If this is not true, the following error message is reported:

```
// Error: Incorrect sharing of pin "/scan_input1": (K15-1)
//        Scan chain "chain1" input
//        Scan chain "chain2" input
```

To support channel sharing between identical and non-identical blocks, this DRC does the following:

- Verifies that the input channels that share the same top-level port do not contain any control channels. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in1": (K15-1)
//          Block "blk1" EDT channel 1 input
//          Block "blk2" EDT channel 1 input
//        When broadcasting channels to non-identical blocks, the
//        shared channels at core level cannot include any EDT
//        control registers.
```

- Verifies that a top-level input port is not broadcast to both an identical and non-identical EDT block. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "core1_edt_channels_in1": (K15-1)
//        Block "blk1" EDT channel 1 input. The input channels of
//        this block are also broadcast to block "blk2".
//        Block "blk3" EDT channel 1 input.
//        When broadcasting channels to identical blocks, those
//        channel ports may not be shared with other non-identical
//        blocks.
//        Block "blk1" is not identical to "blk3" due to differences
//        in scan configuration or embedding.
```

- Verifies that if two non-identical EDT blocks have the same input channel, they also have the same “pulse_edt_before_shift_clocks” signal. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//        inconsistent pulse_edt_before_shift_clocks options:
//        (K15-1)
//        Block "blk1" has pulse_edt_before_shift_clocks set to "on"
//        Block "blk2" has pulse_edt_before_shift_clocks set to
//        "off"
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same number of external pipeline stages. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks with
//        different number of input pipeline stages: (K15-1)
//        Block "blk1" EDT channel 1 input has 4 input pipeline
//        stages
//        Block "blk2"EDT channel 1 input has 5 input pipeline
//        stages
```

- Verifies that if two input channels on two non-identical blocks share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channel1" for blocks of
//        shared input channels with different input channel
//        inversions: (K15-1)
//        Block "blk1" EDT channel 1 input has no input inversion
//        Block "blk2" EDT channel 1 input has input inversion
```

- If an EDT block is inside a core, the tool includes the parent core name of the EDT block in the error message. This example verifies that if two input channels on two non-identical blocks (either from the top-level, from the same core, or from different cores)

share the same top-level port, they must have the same inversions. If this is not true, the following error message is reported:

```
// Error: Invalid sharing of port "edt_channels_in2" for blocks of
//        shared input channels with different input channel
//        inversions: (K15-1)
//        Block "blk1" EDT channel 2 input input has no input
//        inversion.
//        Block "top_core1__unwrapped__blk1" (from core instance
//        "top_core1") EDT channel 2 input input has  input
//        inversion.
//        Block "top_core2__unwrapped__blk1" (from core instance
//        "top_core2") EDT channel 2 input input has  input
//        inversion.
//        Block "top_core2__unwrapped__blk2" (from core instance
//        "top_core2") EDT channel 2 input input has  input
//        inversion.
```

K16

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This rule check verifies that the EDT clock signal has been added as a clock using the add_clocks command.

K17

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This rule check verifies the EDT clock is constrained to its inactive state. This is required in order to avoid disturbing the EDT IP during the capture cycle.

K18

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

This rule check verifies the following three required properties of the test procedure file:

1. Analyzes the off states of the EDT control signals.

   In the IP Creation Phase, Tessent TestKompress creates the EDT IP such that all the control signals are active high. This check verifies that all control signals are, in fact, active high.

Note

If a control signal (EDT clock, for example) is inverted between the chip pin and the EDT IP, and you defined the inversion using the set_edt_pins command in the IP Creation Phase, the signal at the chip level may be active-low.

2. Analyzes the EDT update control signal in the load_unload and shift procedures.

   In order to operate correctly, the EDT update signal is asserted during load_unload before the leading edge of the EDT clock signal. EDT update must be reset after the leading edge of EDT clock in load_unload and before the leading edge of EDT clock in the first shift procedure. It does not matter if EDT update is reset before or after the trailing edge of EDT clock in load_unload.

3. Analyzes the EDT clock signal in the shift procedure.

   This check verifies that the EDT clock is pulsed in the shift procedure in order to decompress vectors provided at the scan channel inputs.

K19

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This design rule check simulates the EDT decompressor netlist by applying pseudorandom patterns and compares the results with the emulated values. If a simulation-emulation mismatch occurs, the DRC performs diagnostic checks to narrow down the number of possible problem sources.

This design rule check also detects missing channel input constraints if a pipeline stage is clocked by a non-EDT clock.

How to Debug K19 Violations

For most common K19 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from channel pins to the decompressor and from the decompressor to the scan chains during shift, it is typically sufficient to use the set_gate_report command to specify Drc_pattern reporting:

set_gate_report *drc_pattern procedure_name*

Then use the report_gates command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K19 rule check as reference points in its automated analysis of K19 mismatches) are reported as part of the K19 error message and provide clues to where the problem may lie.

When you need to see the specific simulation values applied in each shift cycle, you can view the simulated values for all K19-simulated gates, not just the monitor gates by setting the gate report to K19:

set_gate_report *drc_pattern* k19

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

Use “set_gate_report drc_pattern k19” reporting only when necessary. K19 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.

For additional in-depth information on how to reduce the occurrence of K19 violations, refer to the "K19 through K22 DRC Violations" and "Understanding K19 Rule Violations" sections of the *Tessent TestKompress User's Manual*.

K20

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This design rule check identifies the number of pipeline stages within the compactors, based on simulation. It also considers channel output pipelines; reporting any discrepancy between the number of identified and specified pipeline stages between the scan chains and pins (including compactor and channel output pipelines).

Note

If the K19 rule check that verifies the operation of the decompressor fails, then the K20 rule check will not be able to identify the number of pipeline stages.

If this rule check fails, you can still use diagnostic information from the K22 design rule check to help identify a problem. Design rule K22 is checked whether the K20 rule check fails or not.

Note

Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.

K21

Category: EDT

Contexts Supported: dft -edt, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

In the IP Creation Phase, this design rule check verifies that lockup cells will be synthesized if there is a scan chain whose first scan cell captures data at, or after, the rising edge of the EDT clock. The rule check also verifies that lockup cells will be synthesized if the output of the last scan cell changes at, or before, the rising edge of the EDT clock. If bypass logic is included, the rule verifies that lockup cells will be synthesized between the last scan cell of one chain and the first scan cell of another chain when they are connected by the bypass logic, and the output of the last scan cell changes before, or at the same time, the first scan cell captures.

In the Pattern Generation Phase, this rule check verifies the first scan cell of each chain does not capture scan data at the same time the corresponding EDT decompressor output changes its signal. If the EDT compactor is pipelined, the rule check also verifies the compactor does not capture at the same time the output of the last scan cell in each chain changes. This could result in clock skew problems, as the EDT and the core circuitry are part of different clock domains. If a violation is reported, you should add lockup cells or modify the timing. The default handling for this rule violation is warning.The K21 rule check will not be able to identify timing problems in the Pattern Generation Phase if the K19 rule check, which verifies the operation of the decompressor, fails.

K22

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This design rule check simulates the EDT compactor fed by pseudorandom patterns, and compares the results to the emulated values. Two different checks are performed, as follows:

1. The tool applies a pseudorandom pattern to the compactor inputs, with all scan chains enabled.
2. The tool tests each compactor input separately. In this case, one pseudo-random pattern is applied to the selected compactor input, and the compactor is programmed to mask all other inputs (scan chains). Decoder problems and scan chain permutations are detected by this test.

If a simulation-emulation mismatch occurs, the tool automatically performs diagnostic checks to eliminate possible sources of the mismatch. Messages regarding K22 violations usually incorporate relevant information from these diagnostic checks to help you find the source of the mismatch.

Note

Be sure scan channel output pins that are bidirectional are forced to Z at the beginning of the load_unload procedure. Otherwise, the tool is likely to issue a K20 or K22 rule violation, without indicating the reason for the violation.

How to Debug K22 Violations

For most common K22 debug tasks, such as checking for correct values on EDT control signals as well as sensitized paths from the scan chains to the compactor and from the compactor to the channel pins during shift, it is typically sufficient to use the set_gate_report command and specify Drc_pattern reporting:

set_gate_report *drc_pattern procedure_name*

Then use the report_gates command to display simulated values for gates. The positions and values of mismatching monitor gates (certain gates used by the K22 rule check as reference points in its automated analysis of K22 mismatches) are reported as part of the K22 error message and provide clues to where the problem may lie.

Note

In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

When debugging incorrect compactor mask settings for specific patterns simulated by the design rule check, you can view the simulated values for any K22-simulated gate, not just the monitor gates, by setting the gate report to K22:

set_gate_report drc_pattern k22

This is not commonly required, as most problems are usually due to setup problems such as path sensitization and setting of control signals that are usually not pattern-specific or cycle-specific.

Note

Use "set_gate_report drc_pattern k22 reporting" only when necessary. K22 reporting can slow EDT DRC run time and increase memory usage compared to Drc_pattern reporting because the tool has to log simulation data for all simulated setup and shift cycles.

For additional in-depth information on how to reduce the occurrence of K22 violations, refer to the "K19 through K22 DRC Violations" and "Understanding K22 Rule Violations" sections of the *Tessent TestKompress User's Manual*.

K23

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

Description

This design rule check verifies (when the tool generates level-sensitive EDT hardware) that there is no scan chain whose first or last scan cell contains edge-triggered flip flops.

Tessent TestKompress can generate EDT IP based on either edge-triggered flip flops or level-sensitive latches. By default, it generates EDT hardware based on flip flops, with a single clock. The edge-triggered EDT hardware can operate with a design that has scan chains based on edge-triggered scan cells or level-sensitive scan cells. Depending on the clocks used by the first and last scan cells in each chain, the tool generates lockup cells when necessary to prevent clock skew problems.

For pure LSSD designs, the tool can optionally generate EDT hardware based on level-sensitive latches with a master clock and a slave clock. But the level-sensitive EDT hardware requires a design that uses only level-sensitive scan cells, with an appropriate clock scheme for the master and slave clocks. Such a scheme ensures there is no clock skew between the EDT hardware and the first and last scan cells, so the tool does not add any lockup cells.

Note

If you use level-sensitive EDT hardware with bypass logic, you must make sure that scan chain outputs connected to scan chain inputs via bypass logic do not require lockup cells.

K24

Category: EDT

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note, Warning, and Error

report_drc_rules: Not Supported

Description

This design rule check is performed when you have enabled the set_edt_mapping command and are reusing block-level dofiles for top-level pattern creation, or when EDT Finder is enabled. When EDT mapping is enabled, this rule check verifies that the clock, read control, write control, and pin constraint signals specified in the block-level dofiles are consistent with the signals specified at the top level.

Note

K24 does not run when core mapping is used, even if EDT Finder is enabled. When core mapping is used, the connections are automatically extracted to the top and connection mapping issues are reported by other DRCs.

The tool performs this verification by tracing connectivity between specified block-level signals (ports) and primary input or output pins. The tool verifies clock off states and pin constraint values while compensating for inversions in the path. During this rule check, all EDT signals that exist as ports on the block, but not as top-level pins, are updated to refer to the top-level input or output pins as determined by the tracing algorithm.

The K24 rule check outputs three different types of messages depending on the severity level of the results:

- **Note** — K24 outputs an informational note in cases such as when an EDT channel pin name is provided and a pin of that name exists at the top-level but not at the block-level or, in the case where the number of traced pipeline stages along the channel I/O do not match user-specified values.
- **Warning** — K24 issues a warning in cases such as when the traced top-level signal is different from the user-specified signal.
- **Error** — K24 issues an error message in cases such as when an I/O pin or clock port cannot be found.

When EDT Finder is enabled, K24 verifies the existence of the specified channels and maps block-level channels to top-level ports. When EDT Finder is enabled, K24 severity is automatically restored to Error. In this case, you cannot override this severity setting.

For detailed information about creating a top-level dofile for the modular Tessent TestKompress flow, see the Modular Compressed ATPG section of the *Tessent TestKompress User's Manual*.

K25

Category: EDT

Contexts Supported: dft -edt

Default Handling: Error

report_drc_rules: Supported

Description

All block-level EDT pins must be functionally connected in the netlist or have a connection specified during the integration session.

K25 verifies that each block-level EDT pin is either functionally connected in the netlist or has a connection specified. If no connection exists or is specified, an error is returned.

The error message is:

```
//  Error: Update pin in block "B1" is not driven in the netlist and does
not have a defined connection. (K25-2)
```

Procedure Rules (P Rules)

The test procedure file for each scan chain group is checked to ensure adherence to the format rules and accuracy of the test procedure data. All violations of procedure rules are treated as error conditions and cannot be changed—with the exception of rules P30, P31, P32, and P33, which can be ignored.

P1

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Each statement in the test procedure file must have the proper syntax. A syntax error occurs for a statement if there is an incorrect number of arguments or an incorrect ending character (“=” for procedure statements and “;” for all other statements). You can correct this error condition by editing the indicated line of the test procedure file.

The error message is:

```
Error: syntax error near token (Token). (P1)
```

Where Token represents the token in the input file close to where the error occurred.

P2

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

For statements inside a procedure, the time of the statement must not be less than the time of a preceding statement. You can correct this error condition by editing the time value on the indicated line of the test procedure file. The error message is:

```
Time value T less than preceding time value. (P2)
```

Where T is the time defined in the specified statement.

P3

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The P3 rule checks if there is any event in the load_unload procedure at the boundary of the apply shift procedure. You can correct this error by editing the time value on the indicated line of the test procedure file.

The error message is:

```
Line number L, time value not greater than preceding time value for apply
procedure. (P3-1)
```

L is the line number in which the failure occurred.

P4

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

All procedures must end with an **end** statement. To correct this error condition, add an **end** statement at the indicated line of the test procedure file.

The error message is:

```
Line number L, P procedure not ended. (P4-1)
```

L is the line number in which the failure, occurred and P is the procedure name.

P5

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Procedure definitions follow the syntax, **procedure proc_type [proc_name] = …**. For most procedures, the procedure name is optional and it is ignored. Procedures of type seq_transparent, clock, and sub_procedure require a name. This error occurs if one of these procedures is missing its name. Also, this error occurs if the user tries to define more than 32 clock or seq_transparent procedures.

```
Incorrect or missing name for procedure of type P. (P5)
```

Where P represents the procedure type that the error occurred on. Unknown procedure types do not cause a P1 violation.

P6

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may define a procedure only once in a single test procedure file. You can correct this error condition by deleting the duplicated procedure at the indicated line of the test procedure file.

The error message is:

```
Duplicate procedure P.  Original procedure from file F.  (P6)
```

Where F is the file that the first occurrence of the procedure P was found in.

P7

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Statements (except the **procedure** statement) can only execute when a procedure definition is still active. To correct this error condition, add a **procedure** statement prior to the indicated line of the test procedure file.

The error message is:

```
Line number L, no active procedure for S statement. (P7-1)
```

L is the line number in which the failure occurred, and S is the type of statement.

P8

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **load_unload** procedure must contain an **apply shift** statement. To correct this error condition, add an **apply shift** statement at the appropriate place in the **load_unload** procedure of the test procedure file.

The error message is:

```
Line number L, no apply shift in load_unload procedure. (P8-1)
```

L is the line number of the end of the load_unload procedure.

P9

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **shift** procedure must contain a **force_sci** statement. To correct this error condition, add a **force_sci** statement at the appropriate place in the **shift** procedure of the test procedure file.

The error message is:

```
Line number L, no force_sci in shift procedure. (P9-1)
```

L is the line number of the end of the shift procedure.

P10

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **shift** procedure must contain a **measure_sco** statement. To correct this error condition, add a **measure_sco** statement at the appropriate place in the **shift** procedure of the test procedure file.

The error message is:

```
Line number L, no measure_sco in shift procedure. (P10-1)
```

L is the line number of the end of the shift procedure.

P11

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If you define a period for a procedure, then the period time must not be less than the time of the last procedure event. To correct this error condition, increase the period time for the indicated procedure.

The error message is:

```
Period T for P is less than time of last statement.  (P11)
```

T represents the period, and P specifies the name of the procedure or the procedure type if it has no name.

P12

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The pin name argument for a **force** or **pulse** statement must be a valid pin name of a primary input or clock signal.

The occurrence message for a **force** statement is:

```
The following occured at T, incorrect pin name P. (P12-1)
```

Where T represents the token in the input file near to where the error occurred and P is the pin name.

The occurrence message for a **pulse** statement is:

```
Incorrect pin [or path] name, P. (P12-1)
P is not a valid clock pin.
```

Where P is the pin or path name.

Troubleshooting

Edit the pin or path name on the indicated line of the test procedure file.

If the pin or path name refers to a clock signal, the clock must be defined using the add_clocks command.

P13

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

For the **force** statement, you may only use the force values 0, 1, X, and Z. To correct this error condition, edit the force value on the indicated line of the test procedure file.

The error message is:

```
The following occured at T, incorrect force value V. (P13-1)
```

Where T represents the token in the input file near to where the error occurred and V is the incorrect value.

P14

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **force_sci** statement in the **shift** procedure. To correct this error condition, delete the **force_sci** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, force_sci only allowed in the shift procedure.  (P14-1)
```

L is the line number in which the failure occurred.

P15

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **force_sci** statement once in the **shift** procedure. To correct this error condition, delete the **force_sci** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate force_sci statement. (P15-1)
```

L is the line number in which the failure occurred.

P16

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **measure_sco** statement in the **shift** procedure. To correct this error condition, delete the **measure_sco** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, measure_sco only allowed in the shift procedure. (P16-1)
```

L is the line number in which the failure occurred.

P17

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **measure_sco** statement once in the **shift** procedure. To correct this error condition, delete the **measure_sco** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate measure_sco statement. (P17-1)
```

L is the line number in which the failure occurred.

P18

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **apply** statement in the **load_unload** procedure. To correct this error condition, delete the **apply** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, apply only allowed in load_unload procedure.  (P18-1)
```

L is the line number in which the failure occurred.

P19

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You cannot have two **apply shift** statements with a value greater than 1 in the **load_unload** procedure.

The error message is:

```
Line number L, duplicate apply shift statement. (P19-1)
```

L is the line number in which the failure occurred and P19 is the rule ID number.

Note

You can repeat the **apply shift** statement for as many serial shifts as necessary; however, you must specify 1 for the value in the additional **apply shift** statements.

P20

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **apply shadow_control** statement in the **load_unload** procedure. To correct this error condition, select the **apply shadow_control** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, duplicate apply shadow_control statement.  (P20-1)
```

L is the line number in which the failure occurred.

P21

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **apply shadow_control** statement may only be used immediately after the **apply shift** statement. To correct this error condition, move the **apply shadow_control** statement from its current position on the indicated line of the test procedure file to the position following the **apply shift** statement.

The error message is:

```
Line number L, apply shift must precede apply shadow_control.  (P21-1)
```

L is the line number in which the failure occurred.

P22

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You must set the number of repetitions for the **apply shadow_control** statement to 1. To correct this error condition, change the repetition argument of the **apply shadow_control** statement on the indicated line of the test procedure file to the value of 1.

The error message is:

```
Line number L, repetitions for apply shadow_control must be  1. (P22-1)
```

L is the line number in which the failure occurred.

P23

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only use the **apply** statement for the **shift** and **shadow_control** procedures. To correct this error condition, delete the **apply** statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, apply procedure P not allowed. (P23-1)
```

L is the line number in which the failure occurred and P is the procedure name.

P24

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

Description

The only allowed command names are **procedure**, **force**, **force_sci**, **measure_sco**, **apply**, **period**, **initialize**, and **end**. Correct this error condition by editing the statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, incorrect command name C. (P24-1)
```

L is the line number in which the failure occurred and C is the command name.

P25

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

Description

You may only use the **initialize** command at time 0 of the **test_setup** procedure. Correct this error condition by moving the statement, on the indicated line of the test procedure file, to the beginning of the **test_setup** procedure.

The error message is:

```
Line number L, initialize command can only be used at time 0  of
test_setup procedure. (P25-1)
```

L is the line number in which the failure occurred.

P26

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The instance name argument for the **initialize** statement must correspond to at least one primitive-level latch or flip-flop gate instance. Correct this error condition by editing the statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, incorrect instance name N. (P26-1)
```

L is the line number in which the failure occurred and N is the instance name.

P27

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

- The only allowed force values for a clock pin are 0 and 1. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

  The error message is:

  ```
  // Error:  Line number L, clock C may not be forced to a V. (P27-1)
  ```

  L is the line number in which the failure occurred, C is the clock pin name, and V is the incorrect value.

- The only allowed force state for a clock pin is OFF when pulsed in the timeplate. To fix a P27 error where a clock pin with pulse timing is forced to its ON state, create another timeplate that does not pulse the clock and use this timeplate for the procedure or cycle in question.

  The error message is:

  ```
  // The following occurred at line L in file F:
  // Error:  Clock C which is pulsed in timeplate T may not be forced to
  on state. (P27-1)
  ```

  F is the file name, L is the line number in which the failure occurred, C is the clock pin name, and T is the timeplate name.

P28

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The only allowed force value for a write control pin is its defined off-state value, unless it is also defined as a clock. Correct this error condition by changing the force value of the statement on the indicated line of the test procedure file.

The error message is:

```
Line number L, write control W may not be forced to a V.  (P28-1)
```

L is the line number in which the failure occurred, W is the write control pin name, and V is the incorrect value.

P29

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

All clocks must be at their off-state prior to any pattern which places a clock line at an on-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// Error: Line number L, clock C not at off-state prior to clock_on
pattern.(P29-1)
```

L is the line number in which the failure occurred and C is the clock pin name.

P30

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A procedure may not place a clock at its on-state at the same time it forces a non-clock pin to a value or place another clock at its off-state. Correct this error condition by changing force statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// Error: Line number L, clock C cannot be forced on at this time.  (P30-1)
```

L is the line number in which the failure occurred, and C is the clock pin name.

P31

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A procedure may not force a non-clock pin to a value at the same time it forces a clock pin to a value. Correct this error condition by changing **force** statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

The error message is:

```
// Error: Line number L, non-clock pin N cannot be forced at this time.
(P31-1)
```

L is the line number in which the failure occurred, and N is the non-clock pin name.

P32

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A procedure may not place a clock at its off-state at the same time it places another clock at its on-state. Correct this error condition by changing **force** statements prior to and including the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

The error message is:

```
// Error: Line number L, clock pin C cannot be forced off at this  time.
(P32-1)
```

L is the line number on which the failure occurred, and C is the clock pin name.

P33

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

When a pattern places a clock at its off-state, all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state at the same time as the indicated line of the test procedure file. The rules checker ignores this condition if you set the handling to “ignore” with the set_drc_handling command.

Note

Partially overlapping clocks are not supported.

The error message is:

```
// Error: Line number L, clock C not at off-state at end of clock_off
pattern.(P33-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and P33-1 is the rule and violation ID number.

P34

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

At the end of all test procedures (except **test_setup** procedure), all clocks must be at their off-state. Correct this error condition by forcing the indicated clock to its off-state prior to the indicated line of the test procedure file.

The error message is:

```
Line number L, clock C not off at end of P procedure. (P34-1)
```

L is the line number in which the failure occurred, C is the clock pin name, and P is the procedure name.

P35

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

At the end of the **master_observe** and **shadow_observe** procedures, all constrained pins must be at their constrained states. The tools assume they are at that state prior to the procedures. Correct this error condition by forcing the indicated pin to its constrained state prior to the indicated line of the test procedure file.

The error message is:

```
Constrained pin L not at constrained value at end of P  procedure. (P35-1)
```

L is the line number in which the failure occurred, and P is the procedure name.

P36

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The test procedure file must contain a **load_unload** procedure. Correct this error condition by adding a **load_unload** procedure to the test procedure file.

The error message is:

```
No load_unload procedure in group procedure file. (P36-1)
```

P37

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The test procedure file must contain a **shift** procedure. Correct this error condition by adding a **shift** procedure to the test procedure file.

The error message is:

```
No shift procedure in group procedure file. (P37-1)
```

P38

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If the test procedure file contains an **apply shadow_control** statement, the test procedure file must contain a **shadow_control** procedure. Correct this error condition by adding a **shadow_control** procedure to the test procedure file or by deleting the **apply shadow_control** statement.

The error message is:

```
No shadow_control procedure in group procedure file. (P38-1)
```

P39

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If the test procedure file contains a **shadow_control** procedure, the test procedure file must also contain an **apply shadow_control** statement. Correct this error condition by adding an **apply shadow_control** statement or by deleting the **shadow_control** procedure in the test procedure file.

The error message is:

```
Unused shadow_control procedure in group procedure file.  (P39-1)
```

P40

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If you turn on the skew load option, the test procedure file must contain a **skew_load** procedure. Correct this error by adding a **skew_load** procedure to the test procedure file or by turning off the skew load option with the set_skewed_load command.

The error message is:

```
Skewed_load may not be used without a skew_load procedure.  (P40-1)
```

P41

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The values of all pins at the beginning of the **shift** procedure must be the same as at the end of the **shift** procedure. Correct this error condition by changing force statements in the **shift** or **load_unload** procedures.

The error message is:

```
P initial value different from final value in shift  procedure. (P41-1)
```

P is the pin name.

P42

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If there are multiple test procedure files, there can be no more than one **test_setup** procedure. Correct this error condition by deleting the extra **test_setup** procedures.

The error message is:

```
Multiple test_setup procedure defined in test procedure  files. (P42-1)
```

If there are multiple test procedure files, there can be no more than one **test_end** procedure. Correct this error condition by deleting the extra **test_end** procedures.

The error message is:

```
Multiple test_end procedure defined in test procedure  files. (P42)
```

P43

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You must place all write and read control lines at their off-state at time 0 of the **load_unload** procedure. Correct this error condition by adding the appropriate force statements to the test procedure file or by deleting the write or read control lines.

The error message is:

```
T control N not at off-state at time 0 of load_unload  procedure. (P43-1)
```

T is the type of control (write or read) and N is the name of the control line.

P44

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only place the **restore_pis** statement at the end of a **seq_transparent** procedure. Correct this error condition by either removing the statement or by placing it at the end of a valid **seq_transparent** procedure.

The error message is:

```
Line number L, restore_pis can only be used at the end of  the
seq_transparent procedure. (P44-1)
```

L is the test procedure file line number where the error occurred, and P44-1 is the rule and violation ID number.

P45

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

You may only place the **condition** statement at the beginning of a **seq_transparent** procedure. Correct this error condition by either removing the statement or by placing it at the beginning of a valid **seq_transparent** procedure.

The error message is:

```
Line number L, conditions can only be used at time 0 of  seq_transparent
procedures. (P45-1)
```

L is the test procedure file line number where the error occurred, and P45-1 is the rule and violation ID number.

P46

Category: Procedure

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **condition** statement must identify a pin pathname that connects to the output of a scan state element. The path between the two points can only contain buffers and inverters. Correct this error condition by either removing the **condition** statement or by correcting the pin pathname

The error message is:

```
Invalid condition for nonscan cell N (G) during procedure P. (P46-1)
```

N is the name of the non-scan cell, G is the gate ID number, P is the **seq_transparent** procedure name, and P46-1 is the rule and violation ID number.

P47

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The statement mentioned is not allowed to be used in the type of procedure that is named. For example, it is not legal to have a measure statement in a **load_unload** procedure.

The error message is:

```
Statement S can not be used in P procedure. (P47)
```

Where S is the statement, and P is the name of the procedure.

P48

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The set strobe_window time statement (see " Set Statement" in the *Tessent Shell User's Manual*) specifies a bad value for the strobe_window time.

The error message is:

```
Negative or zero strobe window specified. (P48)
```

P49

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The set time scale statement (see " Set Statement" in the *Tessent Shell User's Manual*) specifies a bad time value.

The error message is:

```
Negative or zero value specified for the time scale. (P49)
```

P50

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The pulse width value for the named pulse statement in a named procedure or timeplate is zero or negative. The error message can appear for either timeplates or procedures, and will use either the word “timeplate” or “procedure.”

The error message is:

```
Negative or zero pulse width for S specified in [ timeplate | procedure ]
P. (P50)
```

Where S is the pulse statement, and P is the procedure name.

P51

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The named statement occurs more than once in the named timeplate. For example, forcing the same pin twice in a timeplate is not allowed. Pulsing a clock more than once in a single timeplate is also not allowed.

The error message is:

```
Duplicate S statement in timeplate T. (P51)
```

Where S is the statement that occurs more than once, and T is the timeplate.

P52

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A pulse statement was used with a pin name which is not a clock pin. This message can also appear for either timeplates or procedures.

The error message is:

```
Non-clock pin P cannot be pulsed in [ timeplate | procedure ] S. (P52)
```

Where P is the pin name, and S is the timeplate or procedure.

P53

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The named timeplate does not specify a pulse statement required by one of the procedures which uses this timeplate. For example, if a ram_sequential procedure contains a pulse_read_clock statement, but the timeplate that the procedure uses does not contain a pulse statement for any read clock, this warning will be issued. You can correct this warning by adding pulse statements for all needed clocks in the timeplate definitions.

The error message is:

```
No clock pulse timing in timeplate T which matches event E in procedure P,
two cycle clock pulses will be used. (P53)
```

Where T is the timeplate statement, E is the pulse event, and P is the procedure.

Note

A P53 warning will not cause the Vector Interfaces outputs to create incorrect patterns, just larger patterns. During pattern output, if a timeplate does not contain a needed clock pulse statement, the clock pulse will be created by using two cycles and the force_pi time of the timeplate.

Resolving P53 Errors for analyze_simulation_mismatches

When using the analyze_simulation_mismatches command for a pattern with a P53 violation, the tool issues an error and stops the DRC checking:

```
// Error: P53 violation may produce incorrect "analyze simulation
// mismatches" result.
// Suggest to fix the test procedure timeplate to avoid P53.
```

The problem of the P53 error is due to an undefined clock waveform in the used timeplate.

This violation can be seen when saving patterns as in the following example:

```
// command: write_pattern testcase_failing_patt.v -verilog -serial -all
// -rep
//  Warning: No clock pulse found in timeplate tp_shift which matches
// event /RESETN in procedure used in patterns, two cycle clock pulse
// will be used. (P53)
```

This example shows that the P53 error is issued when saving the Verilog test bench.

The reason for this is the /RESETN is pulsed in some patterns, but the timeplate used for the pattern “tp_shift” does not contain a pulse time for /RESETN, which causes the pattern to expand the capture cycle into two cycles. The P53 rule changes the timing of test patterns and causes an unnecessary increase in test application.

It is recommended you fix any P53 issues first when you want to debug the mismatch issue.

P54

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Some statement has referenced a name that is not defined. For example, using the “timeplate” statement in a procedure to reference a timeplate that has not yet been defined. Or, using the “force” statement on a pin name that the ATPG tool cannot find.

The error message is:

```
Undefined identifier N referenced by S. (P54)
```

Where N is the undefined name and S is the statement referring to N.

P55

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This error message indicates that an old procedure type of statement is used within a cycle-based procedure, or that a cycle-based procedure statement is used in an old style procedure. For example, it would be illegal to put a **break_repeat** statement in a procedure that uses timeplates and cycles; put a force statement with a time value in a cycle-based procedure; or use old procedure statements in a new procedure (for example, the capture procedure).

The error message is:

```
Illegal mix of time based and cycle based statements in procedure P. (P55)
```

Where P is the procedure name.

P56

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The named timeplate or procedure is empty. If it is a procedure, it has no statements, including no timeplate references.

The error message is:

```
[ Timeplate | Procedure ] S has no statements. (P56)
```

Where S is the named timeplate or procedure.

P58

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The time specified for a break_repeat statement is too small to be useful, for example, it is equal to one or less.

The error message is:

```
Break_repeat time value T is too small in procedure P. (P58)
```

Where T is a number, and P is the name of the procedure in which T resides.

P59

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The “set time scale” statement in a procedure file specifies a different time scale then what was already specified in the same file or a previously parsed file. All files must specify the same time scale.

The error message is:

```
New time scale does not match previous time scale. (P59)
```

P62

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

In a definition statement, the identifier N has already been used in a previous definition. For example, if there is a pin name in the design called “GROUP1” and there is an “alias” statement which tries to define an alias called “GROUP1,” the user will receive the following P62 message, “alias name GROUP1 already in use as pin name.”

The error message is:

```
X name N already in use as Y. (P62)
```

Where X is the type of definition, and Y is the previous definition type.

P63

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Only one occurrence of a pin is allowed in an alias statement. Pin P occurs more than once in the current alias statement.

The error message is:

```
Duplicate pin P in alias statement.  (P63)
```

P64

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Pin P has a direction (input, output, bidirectional) that is not compatible with statement S.

```
Direction of pin P is not compatible with S statement.  (P64)
```

For example, the following occurrence message:

```
Direction of pin Out1 is not compatible with force statement. (P64)
```

Out1 is an output pin in the netlist that is not compatible with a force statement.

P65

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A duplicate statement S has been found in procedure P. The tool will ignore the first occurrence of this statement. A duplicate statement is one that occurs at the same time and on the same pin (if applicable) as a previous statement.

For example, if a statement in a procedure causes pin A to be forced to a zero at time zero, and then a subsequent statement causes pin A to be forced to a one at time zero, the user will receive a P65 message and the first force statement will be ignored.

The error message is:

```
Duplicate S statement in procedure P, first statement being ignored. (P65)
```

S is the either just the event type or the event type and pin name when needed. P is the procedure type.

P66

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

The P66 rule is used to check for missing statements within a load_unload_registers procedure. For example, if no Shift block is specified, or if an event statement using the '#' character is not present for each shift_assigment passed to the load_unload_registers procedure, a P66 is issued.

The error message is:

```
Error:  Procedure procedure_name is missing required statement_string
statement. (P66)
```

See "Serial Register Load and Unload for LogicBIST and ATPG" for complete information.

P70

Category: Procedure

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (can be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Source clocks defined in clock control definitions cannot be constrained.

The error message is:

```
// Warning:  Source clock <name> declared in clock control <name> cannot
be constrained. (P70-<occurence>)
```

P71

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

The pin pathname listed in an internal clock definition must be the pin pathname of an existing clock.

This rule verifies that the pin pathname specified by the CLOCK_CONTROL keyword in the clock definition defines an existing clock. Clocks are added with the add_clocks command. Internal clocks are added with the add_clocks -internal command.

The error message is:

```
Error: Name "p1" used in clock control is not defined as a clock. (P71)
```

Where P1 is the pin pathname of the clock.

P72

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

The global control conditions and source clocks defined in multiple clock controls of the same or equivalent clocks must be exactly the same.

If the global controls defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock
controls with different global controls defined for the same clock. (P72)
```

If the source clock list defined in the multiple clock controls are not exactly the same, the following error message is reported.

```
Error: Unable to load clock control for clock "p1" due to multiple clock
controls with different source clocks defined for the same clock. (P72)
```

Where P1 is the pin pathname of the clock.

P73

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

A FORCE statement cannot be used to force an on value on a clock pin. P73 verifies that no clock pins are forced on in clock control definitions.

The error message is:

```
Error: Cannot force clock "p1" to its on value in clock control for clock
"p2".(P73)
```

Where both P1 and P2 are the pin pathnames of the clocks.

P74

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

When multiple clock control definitions are used, compatible conditions must exist to turn off other clock pins while the clock being controlled is pulsed.

Also, a clock pin cannot be forced off when it is defined as a source clock.

This rule check verifies that the clock control conditions defined by multiple clocks are compatible and that the clock being forced to off is not defined as a source clock in the same clock control definition.

If incompatible clock conditions exist, the following error message is reported.

```
Error: Clock “P1” is explicitly forced off in clock control for clock
“P2”. However, there is no way to force “P1” off while pulsing “P2”.(P74)
```

If the clock that is turned off is defined as the source clock in the clock control definition of the equivalent clock, the following error message is reported.

```
Error: Clock “P1” is forced to be off explicitly in clock control for
clock “P2” while it is defined as source clock at the same time. (P74)
```

Where P1 and P2 are the pin pathnames of the clocks.

P75

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

Clock control definitions cannot contain conditions that force the clock being controlled or an equivalent clock to an off state.

If a FORCE statement is used to force a controlled clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for the
same clock.(P75)
```

If a FORCE statement is used to force an equivalent clock off, the following error message is reported.

```
Error: Clock "p1" cannot be forced off when defining clock control for its
equivalent clock "p2".(P75)
```

Where P1 and P2 are the pin pathnames of the clocks.

P76

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

At least one of the following components must be defined in a clock control definition:

- ATPG_CYCLE #, END block
- ATPG_SEQUENCE, END block
- ATPG_SEQUENCE <N> <M>, END block
- FORCE statement that forces the non-clock pin to off
- CONDITION statement

If none of the statements/procedures listed are present, one of the following warnings display:

```
Warning: The clock control for clock “P1” is unconditional for ATPG_CYCLE
"p2".(P76)

Warning: The clock control for clock “P1” is unconditional for
ATPG_SEQUENCE applied to all capture cycles.(P76)

Warning: The clock control for clock “P1” is unconditional for
ATPG_SEQUENCE "p1" "p2". (P76)
```

Where P1 is the pin pathname of a clock and P2 and P3 are capture cycle numbers. The tool assumes that the clock is always pulsed in the specified capture cycles when test patterns are generated.

P77

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

A non-clock pin cannot be defined as a source clock.

This rule check verifies that the pin pathname specified by the SOURCE_CLOCK keyword is a clock pin.

If the pin pathname specified by the SOURCE_CLOCK keyword is not a clock pin, the following error message is reported.

```
Error: Cannot define non-clock pin "p1" as SOURCE_CLOCK. (P77)
```

Where P1 is the pin pathname of the clock.

P78

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

Multiple sequence blocks for a clock must contain local force or condition statements.

If a {ATPG_SEQUENCE, END} or {ATPG_SEQUENCE <N> <M>, END} block is declared without any local force or condition statement, the following error message is reported.

```
Error: Multiple ATPG_SEQUENCE blocks without local condition are defined
in clock control for clock “P1”.(P78)
```

Where P1 is the pin pathname of the clock.

P79

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

A clock control definition containing both sequence and per-cycle blocks must provide mutually exclusive force and/or condition statements to control clock pins.

If an ATPG_SEQUENCE block is declared both with and without force or condition statements, the following error message is reported.

```
Error: ATPG_SEQUENCEs with and without local condition are defined in
clock control for clock “P1”.(P79)
```

If an ATPG_SEQUENCE block is declared without force or condition statements and a ATPG_CYCLE block declares force or condition statements, the following error message is reported.

```
Error: Both ATPG_SEQUENCE mode without local condition and per-cycle mode
with local condition are defined in clock control for clock "p1".(P79)
```

Where P1 is the pin pathname of the clock.

P80

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

Conditions defined to pulse clocks in multiple clock control block definitions must be mutually exclusive.

This rule check verifies that:

- Sequence clock control conditions that pulse the clock are mutually exclusive with the per-cycle clock control conditions that pulse the same clock.
- Sequence clock control conditions that pulse the clock are mutually exclusive of the clock control condition used to pulse the same clock in another sequence definition.

If the clock control condition to pulse a clock in sequence block is not mutually exclusive with the clock control condition for the same clock in a per-cycle block a P80 violation is reported. Depending on the violation, one of the following error messages is reported.

```
Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions
defined
in ATPG_CYCLE P4.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions
defined
in ATPG_SEQUENCE P4 P5.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_SEQUENCE "p2" "p3" are not mutually exclusive with the conditions
defined in ATPG_SEQUENCE applied to all capture cycles.(P80)

Error: In clock control for clock “P1”, the conditions defined in
ATPG_CYCLE "p2" are not mutually exclusive with the conditions defined in
ATPG_SEQUENCE applied to all capture cycles.(P80)
```

Where P1 is pin pathname of the clock and P2, P3, P4, and P5 are capture cycles.

P81

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

There must be a disabled condition defined for sequence clock control definitions, either by applying a clock control condition or setting the source clock to the off state.

This rule check verifies that a clock controlled by a sequence block can be disabled.

If there is no way to disable a clock controlled by a sequence block a P81 DRC violation is reported. Depending on the violation, one of the following warning messages is reported:

```
Warning: No way to disable sequence mode clock control for clock “P1”.
(P81)

Warning: No way to disable sequence mode clock control for clock “P1” by
disabling clock control condition. (P81)
```

Where P1 is the pin pathname of the clock.

P82

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

There must be a way to disable clocks defined with per-cycle clock control.

If there is no way to disable a clock defined with per-cycle clock control by either applying a condition or setting the source clock to the off state, the following warning message is reported:

```
Warning: No way to disable cycle mode clock control for “P1”.(P82)
```

If there is no way to disable a clock defined with per-cycle clock control using a condition, but it can be disabled by setting the source clock to the off state, the following warning message is reported:

```
Warning: No way to disable cycle mode clock control for clock “P1” by
disabling clock control condition.(P82)
```

Where P1 is the pin pathname of the clock.

P83

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

When using per-cycle clock control, there must be a way to disable a clock after the capture cycle.

This rule check verifies the on condition for the clock during the capture cycle is mutually exclusive with the off state condition for the same clock after the capture cycle.

If there is no source clock defined or the source clock cannot be suppressed (defined as free running), one of the following warning messages is reported, and the generated test patterns do not include P1 capture cycles while the clock under control is pulsed at P1.

```
Warning: No way to pulse clock “P1” at capture cycle
"p2" while turning off the same clock after the capture cycle "p2" in
cycle mode clock control for clock “P1”. (P83)
Warning: No way to turn off clock “P1” after capture
cycle "p2" in cycle mode clock control for clock “P1”. (P83)
```

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported.

```
Warning: No way to pulse clock “P1” at capture cycle
"p2" while turning off the same clock after the capture cycle "p2" in
cycle mode clock control for clock “P1” by disabling clock control
condition. (P83)

Warning: No way to turn off clock “P1” after capture
cycle "p2" in cycle mode clock control for clock “P1” by disabling clock
control condition. (P83)
```

Where P1 is the pin pathname of the clock and P2 is the capture cycle.

P84

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

When both sequence mode and cycle mode clock controls are defined for a clock, there must be a way to disable the internal clock for both definitions.

This rule check verifies that a clock running in both sequential and per-cycle mode can be disabled.

If a clock running in both sequence and per-cycle cannot be disabled, the following warning message is reported.

```
Warning: No way to disable both sequence mode and cycle mode clock control
for clock "p1" simultaneously.(P84)
```

Where P1 is the pin pathname of the clock.

P85

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

When clock controls exist for two clocks not defined as equivalent, there must be a way to turn off both clocks simultaneously during a capture cycle.

If both clocks cannot be disabled simultaneously during a capture cycle, the following warning message is reported.

```
Warning: Clock control is unable to turn off clock “P1” and clock “P2”
simultaneously at capture cycle "P3". (P85)
```

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P86

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not supported

Description

When clock controls exist for two clocks that are not defined as equivalent and one clock is pulsed during a capture cycle, there must be a way to turn off the other clock during the same capture cycle.

If there is no source clock defined or all source clocks are defined as free-running the following warning message is reported:

```
Warning: Clock control is unable to turn on clock “P1” while turning off
clock “P2” during capture cycle “P3”. (P86)
```

If it is possible to suppress the clock under control by turning off the source clock, one of the following warning messages is reported:

```
Warning: Clock control is unable to turn on clock “P1” while turning off
clock “P2” during capture cycle “P3” by disabling clock control
condition.(P86)
```

Where P1 and P2 are the pin pathnames of the clocks and P3 is the capture cycle.

P87

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

A clock control cannot be defined for a free running clock.

If a clock control is defined for a free running clock, the following error message is reported.

```
Warning: Cannot define clock control for free running clock “P1”.(P87)
```

Where P1 is the pin pathname of the clock.

P88

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

A CONDITION or FORCE statement cannot assign a value to a non-scan state element.

If a CONDITION or FORCE statement assigns a value to a non-scan state element, the following error message is reported.

```
Error: Value is forced at non-PI and non-scan cell "P1" in clock control
for clock “P2”. (P88)
```

Where P1 is the pin pathname of the gate a value is assigned to and P2 is the name of the clock under control.

P89

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not supported

Description

When you use clock control definitions, all unconstrained internal clocks (defined using the "add_clocks -internal" command) must have an associated clock control definition.

When no clock control is defined for an unconstrained internal clock, the tool displays following error message:

```
Error: Clock control is not defined for the internal clock "P1". (P89)
```

Where P1 is the clock pin pathname.

Note that an unconstrained internal clock does not need a clock control definition and does not cause a P89 error under any of the following circumstances:

- The internal clock is defined as pin-equivalent to other clocks and one of the equivalent clocks has clock control defined.
- The internal clock is a free-running or always-capture clock.
- The internal clock is an asynchronous clock (defined with "add_clocks -period").

P90

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

Internal clock controls cannot be dependant on source clocks defined by other internal clock controls.

For example, when the clock control for clock clk1 defines clk2 as source clock, the clock control for clock clk2 defines clk1 as source clock.

However, you can define the clock under control as the source clock for itself to unconditionally force itself to an off state.

When no clock control is defined for an unconstrained internal clock, the following warning message is reported.

```
Error: Clock controls for clocks “P1” and “P2” depend on each other
through source clocks.(P90)
```

Where P1 and P2 are pin pathnames of clocks.

P91

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not supported

Description

Mutually exclusive control conditions cannot exist for different ATPG_CYCLEs of the same clock.

If mutually exclusive control conditions are defined in different ATPG_CYCLES for the same clock, the following error message is reported:

```
Warning: In clock control for clock “P1”, the conditions defined in
ATPG_CYCLEs "P2" and "P3" are mutually exclusive.
```

Where P1 is the clock pin pathname and P2 and P3 are capture cycles.

P92

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that added internal PI (primary input) signals are controlled deterministically. Specifically, verifies that every internal PI signal satisfies at least one of the following criteria:

- The internal PI has a pin constraint.
- The internal PI has an ATPG constraint.
- The internal PI is a free_running or always_capture clock.
- If NCP are present and enabled, the internal PI must be forced all the time in the internal mode of every enabled NCP.

If none of these criteria is true, the following error message is reported:

```
//Error: Internal primary input /int_clk (42) is not controlled. (P92-1)
```

Note, the following cases are not checked:

- An internal PI that is defined as a scan chain input pin.
- An internal PI that is blocked by pin constraints.
- An internal PI that passes P89 (an internal clock that is controlled by clock control definitions).

You can use the add_input_constraints and add_atpg_constraints commands to properly constrain an internal PI to the right value.

P93

Category: Procedure Rules

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not supported

Description

The clock under control cannot define itself as the source clock. If this is the case, the tool issues following error message:

```
//Error: Clock '<clock_pin_pathname>' cannot be defined as source clock
for itself. (P93)
```

Core Mapping for ATPG and Scan Pattern Retargeting Rules (R Rules)

Prior to performing core mapping for ATPG or for Scan Pattern Retargeting, Tessent Shell performs rule checks on the design as you switch from setup to analysis mode. This design rule checking phase reads the Tessent core description (TCD) files and core instance information.

The tool applies all R rules, except for R7, to the Core mapping for ATPG and Scan Pattern Retargeting functionality.

R1

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting, patterns -failure_mapping

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Verifies that ports defined in the core description match pins at the boundary of core instance(s).

This rule helps detect when a core description is associated with the wrong design module.

R2

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Extracts connectivity between the core-level scan pins and top-level ports, including extraction of pipelining and inversion. Then, validates the ability to shift from top-level ports to the scan inputs on the core boundary, and from the scan output pins on the core boundary to top-level ports.

How to Debug a R2 Violation

To debug R2 violations, examine the shift simulation values to determine why the scan from the core to the top level is blocked using the following command.

set_gate_report drc shift

You can use the report_gates command and the trace capability in DFTVisualizer to analyze the blockage.

You can use the analyze_drc_violation command to debug a given R2 violation graphically.

R3

Category: Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (severity cannot be changed)

report_drc_rules: Supported

Description

Verifies that scan input pins of non-identical cores are not shared, unless broadcasting to multiple instances of the same core type.

- When broadcasting to multiple instances of the same core, the pipelining and inversion for each set of equivalent channels must be identical.
- When patterns are loaded from the chip level, if broadcasting to multiple core instances, they must load identical values at the same time.

Verifies that a top-level pin does not drive multiple channels of the same core.

Verifies that the connection from the top-level pin to identical cores connects to the pins of each core in the same order.

R4

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Validates core-level pin constraints against capture-cycle states at the core instance boundary.

The conditions present at the core boundary during core-level pattern generation must be satisfied for each core instance when the core is instantiated in the design. To verify that the same conditions occur, this DRC validates core-level pin constraints against capture-cycle states at the core instance boundary.

Note

This DRC only validates pin constraints that are specified by the add_input_constraints command.

To perform this validation, the tool simulates the top-level test procedures (test_setup, load_unload, and shift) and input constraints, and compares the stable (constant) values during capture to the pin constraint values at the boundary of each core. The core-level input constraints are stored in the core description file.

You can use the analyze_drc_violation command to debug a given R4 violation graphically.

R5

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning

report_drc_rules: Supported

Description

Validates that the stable values of sequential elements at the end of test_setup at the core level are met after simulating the top-level test_setup procedure. Those are the sequential elements where the “stable_after_setup” value was non-X at the core level.

If the sequential element is no longer present in the design in the retargeting phase due to black boxing or gray boxing of the core, the sequential element's value cannot be validated and will be omitted.

Because the tool cannot determine whether mismatches between the test_setup at the core and the top level matter, this DRC defaults to warning. For example, the state of a core’s TAP instruction register may vary after test_setup at the core versus top level, but this may not matter for the core's scan operation.

You can use the analyze_drc_violation command to debug a given R5 violation graphically.

R6

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Verifies the equivalency of the load_unload and shift procedures at the core and top levels.

This DRC validates simulated top-level load_unload and shift procedures against load_unload and shift procedures at the core level.

R7

Category: Scan Pattern Retargeting

Contexts Supported: patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Verifies immediately before generation of core-level retargetable patterns that all primary inputs (PIs) are controlled deterministically.

If any PI (port) was excluded from automatic constraints as a result of applying the is_excluded_from_ isolation_constraints attribute to it, this DRC verifies that the PI is deterministically controlled and not left for ATPG to justify to different values in different patterns.

Specifically, the tool verifies that every PI signal satisfies at least one of the following criteria:

- The PI has a pin constraint.
- The PI has an ATPG constraint.
- If NCPs are present and enabled, the PI must be forced or pulsed in all cycles of the internal mode of every enabled NCP.

Note, the tool does not check the following PIs:

- An internal PI.
- A PI that is blocked by pins and/or cells whose values are constant during capture.
- A PI that is defined as a free-running clock.
- A PI that is defined as an always-capture clock.

R8

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error

report_drc_rules: Supported

Description

Verifies that the core-level load_unload and shift procedures can be correctly retargeted.

In patterns -scan context, when top-level load_unload and shift procedures are present, R8 additionally checks if core-level procedures can be correctly merged with top-level procedures.

The default setting for this rule is an Error. You can change the setting to be either Warning or Note by using the set_drc_handling command.

The following checks are performed during test procedure retargeting:

- The test procedure is in the core description file. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core-level procedures are missing.
```

- Top-level clock off-states are consistent with those at the core level. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Clock pin 'pinName' is driven by top-level port 'portName'. The
// core-level clock pin has an off-state 'value', but the top-level
// clock port has an off-state 'value'.
```

- The path between the core pin and the top-level pin contains only buffers and inverters. If this is not true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core Pin 'pinName' cannot be traced to the top level. Tracing
// stopped at a gate 'gateName' (gateId).
```

- The core-level clock pin that should be pulsed is instead constantly forced. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level clock pin 'pinName' (core 'coreName', mode
// 'coreMode') is now constantly forced to 'value'.
```

- The core-level pin that should be forced is constantly forced to an opposite value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// The core-level pin 'pinName' (core 'coreName', mode 'coreMode')
// is forced to 'value' in the core-level load_unload procedure,
// but it is now constantly forced to 'value'.
```

- The top-level port that is constrained during load_unload can only be forced to the constraint value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed for
// core instance 'coreInstance' (core 'coreName', mode 'coreMode').
// Core pin 'pinName' is driven by top-level port 'portName'.
// The core-level pin is forced to 'value' in the core-level
// load_unload procedure, but the top-level port is currently
// constrained to 'value' during load_unload.
```

- The top-level port is forced to a different value. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed in
// cycle 'cycleNumber' 1 of the 'procedureName' procedure.
// Top-level port 'portName' has conflicting requirements from the
// existing top-level procedure and the core-level pin(s) it drives:
//
// Top-Level Core-Level Core Pin  Inverted Core        Mode
// --------- ---------- -------   -------  ----------  ----------
// 'value'  'value'     'corePin' Yes      'coreName'  'modeName'
// 'value'  'value'     'corePin'          'coreName'  'modeName'
```

- A clock pulse definition is missing in the timeplate. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' which is driving the following
// core-level pin: 'pinName' (core 'coreName', mode 'coreMode')
// has no pulse definition in timeplate 'timeplateName'.
```

- The top-level clock is forced to its on-state. If this is true, the following error is reported:

```
// Error: load_unload procedure merging failed.
// Top-level clock port 'portName' driving the following core-level
// pin: 'pinName' (core 'coreName', mode 'coreMode') is forced to
// its on-state of 'value'.
```

- The top-level clock is pulsed a different number of times in two cores. If this is true, the following error is reported:

```
// Error: load_unload procedure retargeting failed.
// Top-level clock port 'portName' has conflicting pulse
// requirements from the existing top-level procedure and the
// core-level pin(s) it drives:
//
// Clock Pulses  Core Pin     Core        Mode
// ------------  --------     ----------  ----------
// 'pulseCount'  'pinName'    'coreName'  'coreMode'
// 'pulseCount'  'pinName'    'coreName'  'coreMode'
```

R9

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Description

Verifies during core extraction that a core-level capture clock connects to a matching top-level clock.

This DRC only checks clocks on the core boundary; clocks internal to the core (added using “add_clocks –internal” at the core level) are not validated.

If a core-level clock is constrained off, and it is not pulsed in the core-level test_setup, load_unload, shift, or test_end procedures, capture validation is not needed because the R4 DRC validates all core-level input constraints. If the core-level clock is constrained off and is pulsed in any of these procedures, the tool performs the following checks:

- Verifies that the core-level clock connects to a top-level pin (either a port or cut point defined by the “add_clock -internal” command).
- Verifies that the offstates of the clock and top-level pin match (taking inversion into account).

If a core-level clock is not constrained off and is a free-running clock, the tool does the following:

- Verifies that the core-level clock connects to a top-level pin (either port or cut point defined by “add_clock -internal”).
- Verifies that the top-level pin is defined as a free-running clock.
- If a clock is synchronous (not added with “add_clock -period”), the tool verifies that the off-states match, taking inversion into account. If a clock was added with “add_clock -period”, the tool does not compare off-states.
- If a clock is synchronous, the tool verifies that it is not connected to a top-level asynchronous free-running clock.

If a core-level clock is not constrained off and is not a free-running clock, and is not the capture clock, the tool does not perform any additional R9 checks.

Note

This DRC considers a clock to be a capture clock if the clock is not blocked, is not an internal primary input, is not constrained during capture, is a free running clock or is always-capture, or, if one or more NCPs exists, is pulsed at least once in the active NCPs.

If a core-level clock is not constrained off and is not a free-running clock and is a capture clock, the tool performs the following additional checks:

- Verifies that a core-level clock connects to a top-level pin (either a port or cut point defined by the “add_clock -internal” command).
- Verifies that the offstates of a clock and top-level pin match (taking inversion into account).

R10

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Verifies that top-level ICL is available when any of the core instances being retargeted use IJTAG in their test_setup or test_end procedures.

To correct this error condition, you should use the extract_icl command to extract top-level ICL.

By default, this violation is an error and can be downgraded to warning or note. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the tool bypasses any remaining IJTAG retargeting.

This DRC reports the following possible error messages:

```
//  Error: IJTAG was used in the test_setup and test_end procedures of
//         core 'piccpu_maxlen16' (mode 'internal')
//         but top-level ICL was not extracted or read.
//         Use the 'extract_icl' command to extract top-level ICL or read
//         the top-level ICL model. (R10-1)
//  Error: There was 1 R10 violation (Top-level ICL not extracted while
//         IJTAG used in cores).

//  Note:  IJTAG was used in the test_setup and test_end procedures of
//         core 'piccpu_maxlen16' (mode 'internal')
//         but top-level ICL was not extracted or read.
//         The PDL will not be automatically retargeted and appended to
//         the top-level procedure. (R10-1)
```

R11

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Verifies that every core instance that uses IJTAG at the core-level matches an ICL instance at the top-level.

By default, this violation is an error and can be downgraded to warning or note. If you downgrade this DRC to a lesser violation and the lesser violation occurs, the IJTAG retargeting capability skips any core instance with IJTAG that does not have a corresponding ICL instance at the top level.

This DRC reports the following possible error messages:

```
//  Error: Current design (mode 'internal', ICL module 'piccpu_maxlen16')
//         is expected to have ICL instance 'tessent_occ_i_clk'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-1)
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16')
//         is expected to have ICL instance 'piccpu_maxlen16_edt_i'
//         in the ICL model, but no such ICL instance was found.
//         Read the missing ICL model and rerun ICL extraction. (R11-2)
//  Error: Core instance 'chip_int_1/piccpu_2' (core 'piccpu_maxlen16',
//         mode 'internal') uses IJTAG but has no ICL instance
//         corresponding to it in the ICL model.
//         Rerun ICL extraction and provide the missing ICL model. (R11-3)
//  Error: There were 3 R11 violations (Missing ICL instance for core
//         which uses IJTAG).
```

R12

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Warning (severity can be changed)

report_drc_rules: Supported

Description

Verifies that core-level iCalls are at the end of the test_setup procedure or the start of the test_end procedure.

The tool automatically maps all iCalls located at the end of the test_setup procedure or at the beginning of the test_end procedure; all others are ignored.

By default this violation is a warning and can be changed to error or note. If you downgrade this DRC from error and has a violation, the IJTAG retargeting capability skips all core-level iCalls that have an invalid position.

This DRC reports the following possible error messages:

```
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'setup'
//         of ICL instance 'piccpu_maxlen16_edt_i' (ICL module
//         'piccpu_maxlen16_edt') in test_setup,
//         but not at the end of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-1)
//  Error: Core 'piccpu_maxlen16' (mode 'internal', ICL module
//         'piccpu_maxlen16') calls iProc 'my_proc'
//         in test_end, but not at the beginning of the test procedure.
//         It will not be automatically retargeted and appended to the
//         top-level procedure. (R12-2)
//  Error: There were 2 R12 violations (Position of IJTAG in core-level
//         test procedure prohibits automatic retargeting).
```

R13

Category: Core Mapping for ATPG, Core Mapping for Scan Pattern Retargeting

Contexts Supported: patterns -scan, patterns -scan_retargeting

Default Handling: Error (severity can be changed)

report_drc_rules: Supported

Description

Verifies that the used core-level iProcs are loaded in memory and are registered against the right ICL instances.

By default, this violation is an error and can be downgraded to warning or note. If this DRC is downgraded from an error and has a violation, the IJTAG retargeting functionality skips all core-level iProc calls that have not been read in at the top-level.

This DRC reports the following possible error messages:

```
//  Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'setup'
//         in test_setup, but this iProc is not read for ICL module
//         'piccpu_maxlen16_edt'. (R13-1)
//  Error: Core 'piccpu_maxlen16' (mode 'internal') calls iProc 'my_proc'
//         in test_end, but this iProc is not read for ICL module
//         'piccpu_maxlen16'. (R13-2)
//  Error: There were 2 R13 violations (Missing iProc used at core level).
```

Scannability Rules (S Rules)

Scannability checking ensures that the tool can safely convert a sequential element to a scan element. For each sequential element in the design, the tool performs two main checks. The first check, S1, ensures that when all defined clocks—including sets and resets—are at their off-states, the sequential elements remain stable and inactive. The second check, S2, ensures that for each defined clock, sequential elements can capture data when all the other defined clocks are off. These scanability checks determine if the tool can turn off all set and reset lines, and turn on and off all clock inputs of sequential cells from the design's primary input pins. Without this controllability, the tool will not allow a sequential element to pass scanability checks, which is a requirement to be considered for scan identification.

This checking is similar to the C1 (S1) and C7 (S2) rules checks, which the tool performs to determine the stability of defined scan chains. The C1 and C7 rules perform these same checks on sequential elements that are already converted to scan. For more information on C1 and C7, refer to "Clock Rules (C Rules)" on page 2166.

Once these scanability rules pass, the elements are considered "scannable" and the DRC process treats the non-scan elements as though they were scan elements for the remainder of the DRC rules.

All scanability rules are warnings and, with the exception of S3, their handling cannot be changed with the set_drc_handling command. The default handling of the S3 rule is warning but you can change its handling to other handling types. The following subsections describe the scanability rules and the special handling you can set for them.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates. For more information, see "Reporting Gate Data" on page 2143.

S1

Category: scanability

Contexts Supported: dft -scan

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

Scanability rule S1 checks all the clock inputs (including sets and resets) of each non-scan memory element to ensure that these inputs can be turned off. This rule ensures that non-scan elements that may be converted to scan can be controlled to hold their current data. The report_dft_check command provides information for troubleshooting these failures, by listing the cells that fail this check, the clocks that control them, and their associated gate identification number. You can use this gate identification number with the report_gates command or the **Display** > **Add** menu item in DFTVisualizer to display the failing gate and its associated data.

To debug a S1 DRC Violation without DFTVisualizer, use the command analyze_drc_violation <violation_number>. This will run the DRC analysis for the specific violation and allow the simulation data to be displayed. In the following example, the failure is due to the clock pin “CP” being set to X. To pass the S1 DRC, the clock must be set to its off state of 0. The command sequence noted will work with or without DFTVisualizer.

```
analyze_drc_violation viol S1-2
// Gate report now set to display_sim_data.
// Creating schematic for 3 instances (0 were compacted).
report_drc_rules s1-2
// Warning: Unstable nonscan cell /AA/OUT_2_reg(32) when all clocks are
off. (S1-2)
report_gates 32
// /AA/OUT_2_reg (32) TIEX
// "I0" I (0) 10-
// "I1" I (0) 27-
// CP I (X) 22-/AA/I_1/out
// "I3" I (X) 13-
// "OUT" O (X) 15-
```

scanability rule S1 is a modified version of the C1 clock rule, in that it performs the same type of checking on nonscan sequential elements that C1 performs on scan elements. For additional information on the C1 clock rule, refer to “C1” on page 2170.

Note

Clocks that are defined as free-running cannot be turned off during testing. Therefore, flops driven by free-running clocks fail the S1-rule checking.
Note that Tessent Scan does not use free-running clocks as test-clocks to repair the clock signals of the flops that fail with any S-rules."

S2

Category: scanability

Contexts Supported: dft -scan

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

Scanability rule S2 checks all clock inputs (not including sets and resets) of each non-scan memory element to see whether they can capture data. This rules ensures that a non-scan cell can capture data using one of the defined clocks. In order to be converted to scan, a non-scan cell must be able to capture data when a single clock is on. The report_dft_check command provides information for troubleshooting these failures, by listing the cells that fail this check along with their associated gate identification numbers. You can use the gate identification number with the report_gates command or **Display > Add** in DFTVisualizer to display the failing gate and its associated data.

Scanability rule S2 is a modified version of the C7 clock rule, in that it performs the same type of checking on non-scan sequential elements that C7 performs on scan elements. For additional information on the C7 clock rule, refer to "C7" on page 2195.

S3

Category: scanability

Contexts Supported: dft -scan

Default Handling: Warning

report_drc_rules: Supported

Description

All non-clock primary input pins in the clock cone of non-scan memory elements must be constrained. If these pins are unconstrained and made scannable, trace violations may occur.

Non-clock primary input pins in the clock cone of non-scan memory elements should be constrained in the ATPG dofile or load_unload procedure to correctly sensitize the clock path.

Figure 8-35 illustrates S3 violations where the pin ‘clk’ is defined as the shift clock with an off state value of ‘0’. The non-clock pin ‘en’ causes the violation if it is left un-constrained.

Figure 8-35. Example S3 Rule Violation

The total number of S3 violations are reported in the session transcript during the DRC rule checking. Individual S3 violations can be reported with the report_drc_rules and report_dft_check commands.

The default DRC handling of the S3 rule is “Warning,” which means that memory elements with S3 violations are kept in the pool of scannable candidates. The handling can be changed to “Error” using the set_drc_handling command, in which case the memory elements with S3 violations are excluded from the pool of scannable candidates automatically.

If you want to include some of those memory elements back in scan insertion, you can issue delete_nonscan_instances commands on them. Similarly, if the S3 rule handling is “Warning” (the default), you can obtain the list of the violations using the report_dft_check or report_drc_rules commands and issue add_nonscan_instances commands on a subset of the list explicitly.

Although the handling of the S3 DRC violation can be changed to “Error,” the effect of the violation is different from that of the S1 and S2 rules. The tool does not insert_test_logic on the clock pins to correct an S3 violation even if the user turns on the test logic insertion using the set_test_logic command.

S4

Category: scanability

Contexts Supported: dft -scan

Default Handling: Note

report_drc_rules: Supported

Description

Scanability rule S4 identifies and determines the handling of non-scan memory elements that satisfy all of the following conditions:

- Meet requirements for use as scan elements.
- Maintain constant output values during the shift and capture cycles.
- Are in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.
- Contribute to the data lines of the memory elements in the clock and set/reset paths of other flip-flops/latches, RAMs, and ROMs.

Note

Flip-flop/latch elements that maintain constant output values during shift and capture cycles can potentially disturb the clock and set/reset signals of other flip-flops/latches, RAMs, and ROMs if their constant output values are not maintained.

If non-scan cells that maintain constant output values and are associated with clock and set/reset signals are converted to scan cells, they can prevent the clock and set/reset signal paths from being sensitized properly. When this occurs, more test logic may be required to sensitize the clock path.

You can specify how these non-scan cells are handled by setting S4 to one of the following handling options:

- **Note** — reports the non-scan cells to the log file and leaves them as non-scan cells. Default handling.
- **Warning** — reports the non-scan cells to the log file and converts them to scan cells. The resulting scan cells will have an unknown value (TIEX) for the output during the remainder of the DRC.
- **Ignore** — allows non-scan cells to be converted to scan cells; no reporting is done.

You can report on the S4 results with the report_drc_rules command.

S5

Category: scanability

Contexts Supported: dft -scan

Default Handling: Note (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

Scanability rule S5 identifies clock sources that drive set/reset ports as well as clock ports.

When a clock source (a top-level pin or an internally defined clock net) drives the set/reset port of one or more flip-flops while also driving the clock port of one or more flip-flops, the set/reset port driven by this signal is reported as an S5 violation.

You can optionally fix S5 violations by adding test logic at the set/reset ports driven by the clock. The tool will then treat the set/reset ports as uncontrollable and insert the same test logic that is used to fix the set/reset ports of flip-flops with an S1 violation.

S6

Category: scanability

Contexts Supported: dft -scan

Default Handling: Error

report_drc_rules: Supported

Description

Scanability rule S6 identifies both subchains and pre-existing scan cells (that are not part of declared scan chains or subchains) that have driven scan-in ports and reports them as an S6 violation.

Subchain instances with driven scan-in ports are deleted. That is, they are no longer considered subchains.

By default, an S6 violation is reported as an Error, the system mode remains as setup, and the following error message displays:

```
//  Error: There were <int> scannability rule S6 fails (driven scan-in pin
of a sub chain or of a scan cell in unspecified scan chain).
```

You can optionally choose to change the handling of this rule to Warning. In this case, the tool allows the system mode to change to analysis, but deletes subchain instances that violate the S6 rule and ignores scan cells that violate the S6 rule when stitching into scan chains during the scan chain generation process. The following warning message displays:

```
// Warning: There were <int> scannability rule S6 fails (driven scan-in
pin of a sub chain or of a scan cell in unspecified scan chain).
```

You can use the report_drc_rules command to list the details of an S6 rule violation.

When scan cells produce S6 violations, you can optionally fix them by explicitly declaring the scan chains or subchains that the identified scan cells belong to using add_scan_chains or add_sub_chains commands. The tool will use this information to validate the traceability of declared scan chains or subchains when a test procedure file containing load_unload and shift procedures is present.

You can optionally declare the scan cells violating the S6 rule to be non-scan instances using the add_nonscan_instances command. In this case, the tool ignores these cells when stitching scan chains during the scan chains generation process.

Scan Chain Trace Rules (T Rules)

Using the information in the test procedure files, the rules checker traces the scan chains to identify the scan cells and all memory elements associated with the scan cells. It then classifies the scannable memory elements as either MASTER, SLAVE, SHADOW, COPY, or EXTRA. Violations of scan chain trace rules are error or warning conditions, and you cannot change the handling of these rules—with the exception of rule T18, which you can set to ignore. The following subsections describe each of the trace rules.

Tip: In DFTVisualizer, you can press Ctrl + R to execute a report_gates command on selected gates.

T1

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

All defined scan chains must contain at least one scan cell. Correct this error condition by deleting the indicated scan chains.

The error message is:

```
No scan cells identified in scan chain C. (T1-1)
```

C is the scan chain name.

T2

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

A scannable memory element may not reside in more than one scan chain. You can display the complete paths of the scan chains by using the set_trace_report command by turning trace reporting on and then repeating the rules checking. If you use all scan chains, you may need to make netlist modifications to correct this error condition.

The error message is:

```
N (G) already used in chain trace. (T2-1)
```

N is the instance name, and G is its gate ID number.

T3

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **shift** procedure must create a sensitizable path from the scan chain output back to the scan chain input. An improperly sensitized gate in the scan path will cause an error condition. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. To do this, set the gate reporting to trace, and use the report_gates command for the gate ID number displayed in the error message. This can help you to identify where the blockage occurs; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
Scan chain blocked at gate N (G) after tracing C cells. (T3-1)
```

N is the instance name, G is its gate ID number, and C is the number of scan cells traced in the scan chain.

T4

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A memory element in the scan path must have an active clock during some time period of the **shift** procedure. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the inputs helps you identify how to correct the problem.

The error message is:

```
Clock inputs of N (G) never set active during shift  procedure. (T4-1)
```

N is the instance name and G is its gate ID number.

T5

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

During the **shift** procedure, you must never place an X value on a clock input or an active (X or 1) value on a set or reset input of a memory element in the scan path. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

The error message is:

```
T input of N (G) set to V. (T5-1)
```

T is the type of input (clock, set, or reset), N is the instance name, G is its gate ID number, and V is the invalid state.

T6

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

During any time period of the **shift** procedure, a memory element in the scan path must never have more than one clock input turned on. Correct this error condition by accessing the simulated values of all time periods of the **shift** procedure. You do this by setting the gate reporting to trace and using the report_gates command for the gate ID number displayed in the error message. This can help you identify where the problem occurred; tracing back from the indicated input helps you identify how to correct the problem.

The error message is:

```
Multiple clock inputs of N (G) set active. (T6-1)
```

N is the instance name and G is the gate ID number.

T8

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The **measure_sco** statement in the **shift** procedure must follow the successful observation of the scan cells. To guarantee the observation is successful, the time of the **measure_sco** statement must meet the following conditions:

- It must not occur after exercising the first clock on the memory element closest to the scan chain output.
- It cannot be at time 0 if you capture the data into the last memory element at the end of the **shift** procedure.
- The states on all inputs at the **measure_sco** time must be the same as at the end of the **shift** procedure.

To correct the error condition, you must modify the **shift** procedure to meet the above conditions for the **measure_sco** statement.

The error message is:

```
Invalid measure_sco time. (T8-1)
```

T9

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The traced scan chain input pin must be the same as the scan chain input pin specified with the add_scan_chains command. You can correct this error condition by redefining the scan chain input to be the traced pin.

The error message is:

```
Chain input P1 doesn't match entered value P2. (T9-1)
```

P1 is the name of the traced scan chain input pin, and P2 is the name of the entered scan chain input pin.

T10

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The time of the **force_sci** statement in the **shift** procedure must occur before a clock input of the memory element (closest to the scan chain input) turns on. Correct this error condition by changing the time of the **force_sci** statement to a value less than or equal to the indicated time.

The error message is:

```
Force_sci must occur on or before time T. (T10-1)
```

T is the maximum time.

T11

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A clock input of the memory element (closest to the scan chain input) must not turn on during the **shift** procedure prior to the time of the **force_sci** statement. Correct this error condition by changing the times of the **force_sci** statement or **force** statements.

The error message is:

```
Incorrect propagation of force_sci value to scan cell. (T11-1)
```

T12

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

If a scan cell contains a SLAVE element, the MASTER element is not directly observable.

The error message is:

```
MASTER not observable, a master_observe procedure is required by scan
group G. (T12-1)
```

G is the scan group.

When the handling is set to other than error, the tool automatically makes the necessary MASTER unobservable to prevent a potential simulation mismatch. This applies only to the MASTER of the scan cell containing a SLAVE. The scan cell without a SLAVE retains its original observability. Due to the loss of observability on some MASTERs, test coverage may be reduced. To correct this violation, you can define a master_observe procedure to propagate the MASTER value to the SLAVE.

T13

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If you define and try to use a **master_observe** procedure, there must be at least one scan cell that contains a SLAVE. If there is no such cell, Correct this error condition by deleting the **master_observe** procedure.

The error message is:

```
Master_observe procedure defined but not used. (T13-1)
```

T14

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If you define and try to use a **shadow_control** procedure, there must be at least one identified SHADOW memory element. Correct this error condition by either changing or deleting the **shadow_control** procedure.

The error message is:

```
No SHADOWs identified using shadow_control procedure. (T14-1)
```

T15

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

If you define and try to use a **shadow_observe** procedure, the procedure must observe at least one SHADOW. You can correct this error condition by changing or deleting the **shadow_observe** procedure.

The error message is:

```
No observable SHADOWs identified using shadow_observe  procedure. (T15-1)
```

T16

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

When clocks and write control lines are off and pin constraints are set, the gate that connects to the input of a reconvergent pulse generator sink gate (PGS) in the long path must be at the non-controlling value of the PGS gate.

To correct this error condition, access the simulated values by setting the gate reporting to error_pattern and using the report_gates command. You can also avoid this error by setting the pulse generators to off, but that results in no pulse generator support.

The error message is:

```
Input of pulse generator N (G) not at correct value when clocks are off.
(T16-1)
```

N is the pulse generator instance name, and G is its gate ID number.

T17

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Reconvergent pulse generator sink gates (PGS) cannot connect to any of the following: primary outputs, non-clock inputs of scan memory elements, ROM gates, non-write inputs of RAMs, or transparent latches. To avoid this error, set the pulse generators to off, however note that this results in no pulse generator support.

The error message is:

```
Pulse generator N1 (G1) connected to T N2. (T17-1)
```

N1 is the pulse generator instance name, and G1 is its gate ID number.

T18

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Ignore

report_drc_rules: Not Supported

Description

The maximum number of traced cells in the longest scan chain of a group must equal the entered number of repetitions in the **apply shift** statement in the **load_unload** procedure. The default setting for this DRC is “Ignore.” You can enable this check with the set_drc_handling command.

You can correct this condition by changing the repetition number on the **apply shift** statement. This rules violation has no adverse effects because the tool recalculates the actual number of necessary shifts based on the number of scan cells it encounters.

The warning message is:

```
Traced number of shifts (N1) doesn't match entered value (N2).  (T18-1)
```

N1 is the traced number of shifts, and N2 is the entered number of shifts.

T19

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

Description

If one scan cell has a SLAVE, then all scan cells must have a SLAVE. You must correct this warning by changing the netlist of the scan chains.

The warning message is:

```
N scan cells do not have a SLAVE when some do. (T19-1)
```

N is the number of non-slave scan cells.

T20

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The number of shifts specified using the set_number_shifts command must be at least equal to the length of the longest scan chain. To correct this error set a valid value using the set_number_shifts command.

The error message is:

```
Entered number of shifts N is too small. (T20-1)
```

N is the entered number of shifts, and T20 is the rule ID number.

T21

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The number of independent **shift** applications in the **load_unload** procedure must be less than the scan chain length. Correct this error by removing a sufficient number of independent **shift** applications from the **load_unload** procedure or by deleting the short scan chain.

The error message is:

```
Number of independent shifts N must be less than scan chain  length L.
(T21-1)
```

N is the number of independent **shift** applications, L is the scan chain length, and T21 is the rule ID number.

T22

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If the rules checker traces a scan cell during the application of an independent **shift**, it must also trace that cell during the application of its associated general **shift**. Correct this error by changing the sensitization for either the independent or general **shift,** so that they are sensitizing the same scan cells.

The error message is:

```
N (G) was not used in general chain trace. (T22-1)
```

N is the scan cell instance name, G is its gate ID number, and T22 is the rule ID number.

T23

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The chain length calculated for an independent **shift** must be the same as that calculated for its associated general **shift**. Correct this error by changing the sensitization for either the independent or general **shift,** so that they are sensitizing the same scan cells.

The error message is:

```
Chain length (L1) using independent shift not equal to chain  length (L2).
(T23-1)
```

L1 is the independent shift chain length, L2 is the general shift chain length, and T23 is the rule ID number.

T24

Category: Trace

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

When two adjacent memory elements (source and sink) in a scan chain path are clocked by different shift clocks, the sink must not capture data from the source at the same time the source changes its value. Failure to satisfy this rule can result in unwanted shoot-through during scan chain shifting when clock skew exists between the different shift clocks.

The warning message is:

```
A lockup latch may be required between N1 (G1) of cell M1 and N2 (G2) of
cell M2 in scan chain S. (T24-1)
```

N1 and N2 are the instance names of the adjacent elements clocked by different clocks. G1 and G2 are the corresponding gate IDs. M1(M2) is the cell ID of the scan cell in which N1(N2) is located. S is the name of the scan chain.

Note

If N1(N2) is not within a scan cell, M1(M2) is the cell ID of the scan cell that is driven by N1(N2).

You can correct the violation by inserting a lockup latch between the adjacent memory elements reported in the violation message. To find out which clocks are involved, use the command, report_scan_cells -Range *cell_id1 cell_id2*.

T25

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Scan chain tracing must be independent of whether or not a gate with an X value is initialized to a known value (either 0 or 1). Figure 8-36 is an example of a T25 violation.

Figure 8-36. Example of T25 Violation

In Figure 8-36, the scan path is from DFF gate 25 to PO gate 26. If PI gate 1 is initialized to 1, then the actual scan path is the upper path from gate 25 to gate 26. If PI gate 1 is initialized to 0, then the actual scan path is the lower path from gate 25 to gate 26. In either case, the scan shifting is successful; however, a simulation mismatch could occur in Verilog if PI gate 1 is not initialized (remains an X value). The warning message is:

```
There is a T25 violation. Scan chain tracing has assumed that some gates
with X values are initialized to any non-X values. This is not a problem
in silicon but may cause Verilog simulation mismatches. For more
conservative scan chain tracing, please use the command: "set_drc_handling
-scan_chain_tracing conservative".
```

You can display the T25 violation using the “report drc rule T25” command as shown in the following example:

rep drc rule T25

```
Warning: Scan chain tracing between scan output and scan cell 0 (from
gate 24 back to gate 14) in scan chain 0 assumes that some gates with X
values are initialized to any non-X values. (T25-1)
```

You can debug T25 violations by displaying the gate value in scan shift. To do this, use the set_gate_report command to set gate reporting to drc shift and use the report_gates command with the gate ID number displayed in the message.

The default handling for this rule violation is warning. If you do not want scan chain tracing to pass with a T25 violation, use the “set_drc_handling -scan_chain_tracing conservative” command. For more information, see the set_drc_handling command.

T26

Category: Trace

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

The shift procedure must create a sensitizable path through bidirectional pins at the scan chain output or EDT channel output. The improperly sensitized bidirectional pin causes an error condition.

For scan chain output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional scan chain output pin N in
chain "C". The pin is not forced to Z during measure_sco. (T26-1)
```

For EDT channel output, the tool issues the following error message:

```
Scan chain tracing failed at bidirectional channel output pin N in block
"B". The pin is not forced to Z during measure_sco. (T26-2)
```

N is the pin name, C is the corresponding chain name, and B is the corresponding EDT block name.

You can avoid this rule violation by adding a force to Z for the reported pins in the shift procedure before measure_sco. Alternatively, you can force all bidirectional pins with a ‘force _ALL_BIDI Z’ in the shift procedure before measure_sco. The _ALL_BIDI is a macro that selects all primary bidirectional pins.

You can also define the pin as an output pin instead of as an inout pin in your Verilog file. By doing this, then there is no need to force this pin to Z anymore.

Power-Aware Rules (V Rules)

The DRC rules checked for power-aware designs are described here. You must load power data (using the read_cpf or read_upfcommands) before using the report_drc_rules command for the V DRC rules.

When using the power-aware DRC rules, the following conditions apply:

- **Rule V1 to V7** — Checked after reading a power data file.
- **Rule V8 to V21** — Checked when switching from setup mode to a non-Setup mode.

The tool displays a summary of the V rules when you load power data into the tool without any parsing errors as shown here:

read_upf my_design.upf
report_drc_rules –summary

```
…
T26: #fails=0  handling=error  (bidirectional chain / channel outputs not
forced to Z during chain tracing)
V1: #fails=0  handling=ignore/verbose (no power modes are defined)
V2: #fails=0  handling=ignore/verbose (no default power modes are defined)
V3: #fails=0  handling=error  (multiple default power modes are defined)
V4: #fails=0  handling=error  (no default power domains are defined)
V5: #fails=0  handling=error  (multiple default power domains are defined)
V6: #fails=0  handling=warning (no restore_edge or default_restore_edge is
defined)
V7: #fails=0  handling=warning  (power mode is not reachable)
V8: #fails=0  handling=error  (scan data path is in a power-off domain)
V9: #fails=0  handling=error  (scan control path is in a power-off domain)
V10: #fails=0  handling=error  (power start/end/shutoff condition is X
during the scan shift period)
V11: #fails=0  handling=error  (active power mode has changed during the
scan shift period)
V12: #fails=0  handling=warning  (active power mode may change during the
capture cycle)
V13: #fails=0  handling=warning  (path crosses power domains without a
level-shifter cell)
V14: #fails=0  handling=warning  (path crosses power domains without an
isolation cell)
V15: #fails=0  handling=note  (scan chain contains both retention and
regular cells)
V16: #fails=0  handling=note  (scan chain contains both always-on and
other types of cells)
V17: #fails=0  handling=note  (EDT channel contains both retention and
regular cells)
V18: #fails=0  handling=warning  (power control signals are in a domain
that could be powered-off)
V19: #fails=0  handling=warning  (retention save signal is not off during
shift)
V20: #fails=0  handling=note  (non-scan retention cells are identified in
the design)
V21: #fails=0  handling=error  (power data assertions are detected)
```

V1

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

No power modes are defined when there are multiple power domains. You cannot change the handling.

The violation message is:

```
At least one power mode is required when multiple power domains are
defined (V1).
```

Because power modes in UPF are implicitly learned through power switches control logic, the V1 rule is ignored in UPF format.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V2

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

No default power modes are defined when there are multiple power modes. You cannot change the handling.

The violation message is:

```
A default power mode is required when multiple power modes are defined
(V2).
```

The V2 rule is ignored for UPF format.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V3

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

Multiple default power modes are defined. You cannot change the handling.

The violation message is:

```
Multiple default power modes are defined (V3).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line containing the error.

V4

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

No default power domain is defined when there are multiple power domains. You cannot change the handling.

The violation message is:

```
A default power domain is required when multiple power domains are defined
(V4).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window.

V5

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

Multiple default power domains are defined. You cannot change the handling.

The violation message is:

```
Multiple default power domains are defined (V5).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line containing the error.

V6

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Either restore_edge or default_restore_edge must be defined if the design contains any retention cells; if this rule is violated, the state retention rule is ignored and no retention value is restored.

Note that missing both save_edge and default_save_edge is allowed on CPF as long as restore_edge or default_restore_edge is defined. In this case, the inversion of restore_edge (or default_restore_edge when restore_edge is missing) will be used as save_edge.

The violation message is:

```
No restore_edge is defined for the state retention rule name. (V6-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the retention definition lines.

V7

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A defined power mode is not reachable from the default power mode with the defined power transition rules.

The violation message is:

```
Power mode name1 is not reachable from the default power mode name2 using
any defined power transition rule (V7-n).
```

The V7 rule is ignored for UPF format.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file in a Text Editor window and highlights the line defining the power mode.

V8

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

The scan tracing should not propagate to a scan cell in a power domain that is in the shutoff state. Note that the number of power OFF gates in a given scan chain can be large, so only one V8 violation is recorded per scan chain and the first scan path gate from the sco pin is reported. Violation of the V8 rule can be due to at least one of the following reasons:

- The power mode definition is incorrect.
- The shutoff condition of the power domain is defined incorrectly.
- The test procedures (for example, load_unload or shift) for the scan chain operation are defined incorrectly.
- The inserted scan logic of the design does not consider the power mode properly.

The violation message is:

```
The scan path of chain chain-name is in power off domain domain-name (the
first power off scan path gate is N1 (G1) after tracing C cells ) (V8-n)
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer displays the scan cell in the Debug window with a callout box attached containing the following message:

```
Power Domain: < name>, Power Mode (CPF) or Power State: <name>
```

V9

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

All control logic for the scan operation must be in a power ON domain during the entire scan shift operation. Some popular control logic examples are listed below:

- If EDT logic exists in the design, it should stay at power ON mode during the scan test when the compression mode is used; otherwise the scan operation will fail.
- If the scan_enable signal is derived from a JTAG controller, the JTAG controller should stay at power ON mode during the scan test.
- The shift clock control logic should remain at power ON mode during the scan test. The tool performs the check by monitoring all test procedures except test_setup and ensures that the power domains where the scan control logic are located remain at the power ON mode after test_setup procedure.

The violation message is:

```
The scan chain chain-name is blocked at N1 (G1) after tracing C cells due
to the control gate N2 (G2) in power off domain domain_name (V9-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer displays the control gates in the Debug window with a callout box attached containing the following message:

```
Power Domain: < name>, Power Mode (CPF) or Power State: <name>
```

V10

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

The expression used by mode_transition rule and shutoff condition should not be X during the scan shift period. The tool performs the rule by checking all events to ensure that no expression can be X during the scan shift operation. The expressions to be checked include the shutoff condition of every power domain and the start and end conditions of every mode_transition rule.

The violation messages are:

Case 1 — When shutoff condition of a power domain is X:

```
The shutoff condition of power domain name is X at time t of procedure
procedure-name (V10-n).
```

Case 2 — When the start condition of a mode_transition rule is X:

```
The start condition of mode transition rule name is X at time t of
procedure procedure-name (V10-n).
```

Case 3 — When the end condition of a mode_transition rule is X:

```
The end condition of mode transition rule name is X at time t of procedure
procedure-name (V10-n).
```

The handling of X is different based on the handling of the V10 DRC rule. When the handling of the rule is set to warning, the condition of X will be treated as HI as if the condition is satisfied. When the handling is Note, the condition of X will be treated as LO as if the condition is not satisfied. Note, the violation of this rule may cause the design to be in the wrong power mode and thereby cause a simulation mismatch.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V11

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

During scan chain shift operation, the design should not change power modes because this can cause some power domains to be switched between power OFF mode and power ON mode. This occurs because changing active power domains during the shift period may cause the scan chain configuration to be changed. The tool performs the rule by checking the shutoff expressions of all power domains to ensure they remain false during the scan shift period.

The violation messages are:

Case 1 — When mode transition rules are defined:

```
Active power mode changed from mode1 to mode2 at time t of procedure
procedure_name (V11-n).
```

Case 2 — When mode transition rules are not defined:

```
The power domain name changed from state-1 to state-2 at time t of
procedure procedure_name (V11-n).
```

When the handling of the violation is not error, the scan shift operation may be unreliable and this can result in a simulation mismatch. Note that a mismatch may not be seen in the parallel test bench and this can make the debugging process more complicated.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V12

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Note

Scan insertion in the dft -scan context supports this rule if the design contains pre-existing scan chains and a valid test procedure file.

ATPG should not create_patterns that can turn off an active power domain during the capture cycle; turning off an active power domain during the capture period changes the scan chain configuration. The tool performs the rule by checking on active power domains with shutoff expressions and ensuring they cannot be true during the capture period.

The violation messages are:

Case 1 — When mode transition rules are defined:

```
The power mode may change from name1 to name1 during the capture cycle
(V12-n).
```

Case 2 — When mode transition rules are not defined:

```
The power domain name may change state during the capture cycle (V12-n).
```

When the shutoff condition is true and the handling of the violation is not error, the capture value of the scan cells in the shutoff power domain remains the same as if the power domain is still active. This may cause some test patterns to fail in silicon that may not be detected during the logic simulation phase.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. DFTVisualizer highlights the line in the CPF file that defines the power domain (if applicable) and also the line in the test procedure file that defines the procedure name.

V13

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

A path should not cross different power domains without a level-shifter cell if the power domains can operate in different operation voltages. The following situations can cause a V13 DRC violation:

- An instance is driven by another instance of a different voltage power domain.
- The scan_in pin or the EDT decompressor to the first scan cell crosses different voltage domains.
- The last scan cell to the scan_out pin or EDT compactor crosses different voltage domains.
- The scan cell (i+1) to scan cell i crosses different voltage domains.

The violation messages are:

Case 1 — When violation is on a non-scan path:

```
Path from gate N1 (G1) to gate N2 (G2) crosses different voltage domains
without a level-shifter cell (V13-n).
```

Case 2 — When violation is on a scan path:

```
Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan
chain chain-name crosses different voltage domains without a level-shifter
cell (V13-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer displays the path between the two gates reported in the message in the Debug window and displays the simulation data as a DRC in the Debug window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

V14

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

If a path crosses two power domains and the driving power domain contains the shutoff condition, the path must include an isolation cell. Similar to rule V13, there are four situations to be considered for this rule.

The violation messages are:

Case 1 — When violation is on a non-scan path:

```
Path from gate N1 (G1) to gate N2 (G2) crosses different power domains
without an isolation cell (V14-n).
```

Case 2 — When violation is on a scan path:

```
Scan path from gate N1 (G1) of cell (i+1) to gate N2 (G2) of cell i of scan
chain chain-name crosses different power domains without an isolation cell
(V14-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer displays the path between the two gates reported in the message in the Debug window and displays the simulation data as a DRC in the Debug window. If possible, a callout box for the DRC is attached to the source and the sink containing the power domain names.

V15

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Description

A scan chain should not include both retention cells and regular scan cells. A violation of this rule will make retention test less efficient.

The violation message is:

```
Scan chain chain-name contains both retention cells and regular cells
(V15-n).
```

This violation is not supported in DFTVisualizer.

V16

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Description

Typically, always-on cells are used to program power mode and retention mode. To make the control of the power mode more efficient, a scan chain should not include both always-on cells and other types of scan cells.

The violation message is:

```
Scan chain chain-name contains both always-on cells and other type of
cells (V16-n).
```

This violation is not supported in DFTVisualizer.

V17

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Description

An EDT scan channel should try to prevent the mix of scan chain with retention cells and, regular scan chain without retention cells. This rule prevents the retention cells from being masked by regular cells when the circuit is restored from the retention mode.

The violation message is:

```
Scan channel channel-name contains both retention scan chain and regular
scan chain (V17-n).
```

This violation is not supported in DFTVisualizer.

V18

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

All power control signals, including retention save, retention restore, isolation_enable, and power-domain shutoff conditions should be from always-on power domains that cannot be turned off.

The violation message is:

```
Shutoff condition condition_expression  for domain domain_name  is in non-
always-on power domain (V18-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and highlights the line that defines the power domain.

V19

Category: Power Aware

Contexts Supported: patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Retention save signals should be off during the scan shift period to prevent the loading value from being overridden by the retention saved state.

The violation message is:

```
Save signal condition_expression  for retention rule rule_name  is not off
during the scan shift period (V19-n).
```

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and highlights the line that defines the rule name.

V20

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Note

report_drc_rules: Supported

Description

Only scanned retention cells are supported by the retention test to test the retention functionality. So this rule checks if there are any non-scan retention cells.

The violation message is:

```
Retention cell cell_name is a non-scan cell (V20-n).
```

This violation is not supported in DFTVisualizer.

V21

Category: Power Aware

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

The design should never enter any illegal power configurations specified in CPF command assert_illegal_domain_configurations during any test cycle.

The violation message is:

```
The active power mode violates assert_illegal_domain_configurations name
at time t of procedure proc_name (V21-n).
```

Note

CPF and UPF power data parsers support the Tcl scripting mechanism. The violation of the Tcl format results in an error. Note, the format of the error message is not described in this document.

When you issue the analyze_drc_violation command on an occurrence of this violation, DFTVisualizer opens the CPF file and the test procedure files (if applicable) in a Text Editor window. In the CPF file, DFTVisualizer highlights the assert_illegal_domain_configurations command (if applicable).

Timing Rules (W Rules)

These rules apply to timeplates and the mapping of timeplates to procedures. This includes making sure that the order of events in a procedure is not changed after leaving setup mode. For timing rules specific to the enhanced procedure file, see " Test Procedure File" in the *Tessent Shell User's Manual*.

W1

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This message can only occur when not in setup mode. After leaving setup mode, it is possible to load an enhanced procedure file using the read_procfile command. It is also possible for the enhanced procedure file to load a new procedure which overwrites a procedure already loaded. This is fine as long as the event order in the new procedure matches that of the old procedure. If the new procedure has additional events, the tool will issue this error message:

```
New procedure P has more events than existing procedure. (W1)
```

P is the name of the new procedure.

W2

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Similar to rule W1, but this rule is violated if the new procedure has less events than the old procedure.

The error message is:

```
New procedure P has fewer events than existing procedure. (W2)
```

P is the name of the new procedure.

W3

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Similar to rule W1, this rule is violated if the new procedure has a different event order than the old procedure.

The error message is:

```
New procedure P has a different event order then existing procedure. (W3)
```

P is the name of the new procedure.

W4

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This message is similar to rule W1. Instead of an entire new procedure being loaded, the enhanced procedure file only specifies that a new timeplate is applied to an existing procedure. This is done by specifying a new procedure in the enhanced procedure file with only a timeplate reference statement. If that new timeplate causes the event order in the old procedure to change, the tool will issue this error message:

```
New timeplate T changes event order in procedure. (W4)
```

T is the name of the new timeplate.

W5

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A new non-scan procedure has an event statement that either occurs in the wrong order, or is not allowed in that type of procedure. All non-scan procedures must conform to the event order that is stated for them in " Non-Scan Procedures" in the *Tessent Shell User's Manual*. An example of this would be a capture procedure that has a "measure_po" statement before the "force_pi" statement.

The error message is:

```
Procedure P has an illegal event statement or event order S. (W5)
```

P is the procedure name and S is the illegal statement or event order.

W6

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

If one shift procedure has the “measure_sco” statement occurring after the pulse of the shift clock, then all shift procedures must place the “measure_sco” statement after the pulse of the shift clock, and all load_unload procedures must have a “measure_sco” statement occurring at the end of the cycle right before the “apply shift” statement. Placing the “measure_sco” statement after the pulse statement enables end measure mode which affects the way the Vector Interfaces code writes out parallel load scan patterns. See the “Creating Test Procedure Files for End Measure Mode” section of the *Tessent Shell User’s Manual*.

The error message is:

```
Shift procedure P [does | does not] use end measure, while previous ones
[do | do not]. (W6)
```

P is the procedure name.

W7

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

No timing information has been specified for the named procedure. Either the procedure needs to reference a timeplate, or time values must be associated with the event statements.

The error message is:

```
No timeplate or times specified for procedure P. (W7)
```

P is the procedure name.

W8

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A scan procedure (shift, load_unload, …) has been read in by the read_procfile command, but no “scan_group” statement is in the procedure.

The error message is:

```
No scan group specified for procedure P. (W8)
```

P is the procedure name.

W9

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The named procedure has no event statements, no apply statements, and no timeplate references.

The error message is:

```
No events in procedure P. (W9)
```

P is the procedure name.

W10

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This warning is issued during DRC checking. It indicates that none of the procedure files specified in the add_scan_groups commands have any procedures in them.

The warning message is:

```
No test procedures have been loaded. (W10)
```

W11

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A cycle in a procedure that needs to be split into two cycles in order to use the timeplate that is specified for that procedure. This could occur, for example, if the test_setup procedure contains a pulse statement on a specific clock pin, and yet the timeplate used referenced by this procedure does not contain a pulse statement for the clock. The user will receive a P53 rule message and then a W11 rule message to indicate that the cycle (where the pulse statement is) was split into two cycles in order to create the clock pulse.

The warning message is:

```
Cycle S being split in procedure P. (W11)
```

S in the cycle and P is the procedure name.

W12

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

A timeplate was loaded that is missing a required statement. For example, a timeplate must have a “force_pi” statement. If a required clock pulse statement is missing from the loaded timeplate, you will receive a P53 rule message.

The error message is:

```
No S statement in timeplate T. (W12)
```

S is the statement name and T is the timeplate.

W13

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

The event order for the name procedure has changed while parsing an procedure file, but since the tool is still in setup mode, this change of event order is acceptable. This could happen if two procedure files in two different add_scan_groups commands both have the same procedure (test_setup, for example), but one specifies a different event order than the other. The last one loaded is what will be used.

The warning message is:

```
Event order has changed in procedure P. (W13)
```

P is the procedure name.

W14

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This message indicates that a procedure has been replaced by a new procedure with the same name, same type, and same scan group if applicable. This message indicates that the replacement did not violate any rules. Both procedures have the exact same event order, but they might use different timeplates.

The warning message is:

```
Procedure P replaces same procedure from file S. (W14)
```

P is the procedure name and S is the filename.

W15

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

End measure mode is enabled but an illegal force or pulse statement occurs between the measure_sco statement and the end of the procedure in a shift procedure, or the illegal statement occurs between the measure_sco statement and the apply shift statement in a load_unload procedure. See the "Creating Test Procedure Files for End Measure Mode" section of the *Tessent Shell User's Manual*.

The error message is:

```
Statement S can not follow measure_sco in end measure procedure P. (W15)
```

S is the illegal statement and P is the procedure name.

W17

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Clock_po procedures are used to hold clock pins at certain values. In order to obtain the correct fault coverage, it is important that the timeplates used in a clock_po procedure do not pulse the clock pins, but only force them. If this is not corrected before patterns are saved, the Vector Interfaces code places X values on the output pins for the clock_po procedure and fault coverage decreases.

The error message is:

```
Timeplate T has clock pulses but is used in a clock_po procedure.  (W17)
```

T specifies which timeplate has the clock pulses. To correct this, make sure you define a different timeplate that does not have clock pulses and use this in the clock_po procedure.

W18

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

Many ASIC vendors require that shift procedures only be one cycle in length, and it is very unusual to encounter a multiple cycle shift procedure. This warning alerts the user that something might not be specified correctly in the shift procedure, such as forcing the same pin with more than one value, or pulsing a clock twice.

The warning message is:

```
Shift procedure has more than one cycle, please check if shift clocks are
defined properly. (W18)
```

W19

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Not Supported

Description

This message occurs when using the offstate option improperly. The offstate option is only supported when using the Enhanced Vector Interfaces. Also, the pin referenced with the offstate option cannot be used in a capture, ram_sequential, or clock_sequential procedure. These are considered non-scan procedures.

Any timeplate that uses the offstate statement will be treated as a complex timeplate and will not be allowed to be used in a non-scan procedure. Complex timeplates are most useful in shift procedures where a non-clock pin needs to be pulsed while still maintaining a single cycle in the shift procedure. If you attempt to pulse a signal specified with an offstate option in a capture or other non-scan procedure, you will encounter the following error message.

```
Timeplate gen_tp1 with complex waveforms cannot be used in non-scan
procedure capture.  (W19)
```

W20

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This error occurs if, in a capture procedure, a force_pi event does not occur before the first clock pulse.

The error message is:

```
A force_pi statement does not exist before the first clock pulse in
Procedure P. (W20)
```

P is the procedure name.

To correct this violation, add a force_pi statement to the beginning cycle of the procedure, before the first clock pulse. For a named capture procedure with internal and external modes, add a force_pi statement to the beginning cycle in each mode.

W21

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

In a named capture procedure, a measure_po statement is allowed only in the last cycle of the internal mode. This error occurs if a measure_po statement is found in a cycle where it is not permitted.

The error message is:

```
Procedure P has one or more measure_po statements which do not occur in
the cycle before the last clock pulse. (W21)
```

P is the procedure name.

To correct this violation, remove the measure_po statements from the cycles where they are not allowed.

W22

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

In a named capture procedure, a measure_po statement is allowed only in the last “cyclized” ATPG cycle of the internal mode—before the last clock pulse. This warning alerts you that the tool did not find a measure_po statement in this location. A capture procedure with this warning will not use any PO as a capture point.

The warning message is:

```
Procedure P has no measure_po statement in the cycle before the last clock
pulse. (W22)
```

P is the procedure name.

To view the cyclized information for the procedure, use the report_capture_procedures command.

W23

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This warning alerts you that the external mode's period is longer than the internal mode's period.

The warning message is:

```
The total period for the external mode is longer than the total period for
internal mode in procedure P. (W23)
```

P is the procedure name.

To eliminate this warning, modify the timeplate(s) so that the external mode period matches the internal mode period.

W24

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

In a named capture procedure, the total period of the external mode must match the total period of the internal mode. This error occurs if the internal mode's period is longer than the external mode's period.

The error message is:

```
The total period for the internal mode is longer than the total period for
external mode in procedure P. (W24)
```

P is the procedure name.

To correct this violation, modify the timeplate(s) so that the external mode period matches the internal mode period.

W25

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

Event times in the external and internal modes of a named capture procedure should match. This warning alerts you that an event in the external mode does not have a matching event in the internal mode.

The warning message is:

```
// Warning: External event pulse S at time T needs to have a matching
internal event in procedure P or the signal needs to be disconnected from
internal sequential elements or observe points. (W25-1)
```

Where S is the signal name, T is the time of the event, and P is the procedure name.

In Tessent Diagnosis, these violations if uncorrected can cause pattern verification errors in the tool. See the *Tessent Diagnosis User's Manual* for more information.

W26

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Event times in the external and internal modes of a named capture procedure must match. This error occurs if an event (other than an internal pin event) in the internal mode does not have a matching event in the external mode.

The error message is:

```
Event E N at time T in internal mode does not have matching event in
external mode in procedure P. (W26)
```

E is the type of event, N is the name of the pin to which the event applies, T is the time of the event, and P is the procedure name.

To correct this violation, you need to do two things:

- Add the missing event to the external mode.
- Modify the timeplate(s) so that the event added to the external mode occurs at the same time as in the internal mode.

W27

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

In a named capture procedure, the start time for the scan load cycle in the internal mode must have a match in the external mode. This error occurs if there is not a match in the external mode.

The error message is:

```
Start time for scan load cycle in internal mode does not have a match in
external mode in procedure P. (W27)
```

P is the procedure name.

To correct this violation, adjust the timeplate(s) or procedure so that the internal scan load cycle has a corresponding external cycle starting at the same time.

W28

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

In a named capture procedure, the measure_po time in the external mode must match the internal mode. This error occurs if the measure_po time in the external mode does not match the internal mode.

The error message is:

```
Measure_po time in external mode does not match internal mode in procedure
P. (W28)
```

P is the procedure name.

To troubleshoot this violation, check that there is a measure_po statement in both the internal mode and the external mode. If a measure_po exists in both modes, adjust the timeplate(s) to ensure the measure_po occurs at the same time in both modes.

W29

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

In a named capture procedure, the force_pi time in the external mode must match the internal mode. This error occurs if the force_pi time in the external mode does not match the internal mode.

The error message is:

```
Force_pi time in external mode does not match internal mode in procedure
P. (W29)
```

P is the procedure name.

To troubleshoot this violation, check that there is a force_pi statement in both the internal mode and the external mode. If a force_pi exists in both modes, adjust the timeplate(s) to ensure the force occurs at the same time in both modes.

W30

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Supported

Description

In a named capture procedure, an event on an internal pin is not allowed in the external mode. This warning alerts you that there is an internal pin event in the external mode of the procedure.

The warning message is:

```
Event E N on internal pin is not allowed in external mode of procedure P.
(W30)
```

E is the type of event, N is the name of the pin to which the event applies, and P is the procedure name.

The tool will ignore the internal pin event in the external mode; however, you can avoid the violation by removing the internal pin event from the external mode.

W31

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

A named capture procedure that has an external mode must also have an internal mode. This error occurs when an external mode exists without an accompanying internal mode.

The error message is:

```
Procedure P has an external mode but is missing an internal mode. (W31)
```

P is the procedure name.

To correct this violation, do one of the following:

- Add an internal mode definition to the named capture procedure, or
- Remove the “mode external =” statement and its corresponding “end;” statement from the named capture procedure.

W32

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Not Supported

Description

A force_pi in a named capture procedure must not occur when a clock is in an on state.

W32 ensures the procedure file is valid.

This DRC checks the force_pi statement in named capture procedures in the procedure file when loading a new procedure file, which can occur during DRC, when the read_procfile command is used, or when the write_patterns command is used with a new procedure file specified on the write_patterns command line.

Messaging

If the procedure file fails W32, the following message displays:

```
A force_pi statement occurs while a clock or control is active in
procedure P (W32).
```

Where P is the name of the procedure file.

Troubleshooting

To correct this violation, edit the procedure file so the force_pi statement does not occur during the time that a clock is in its on state.

W33

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Warning

report_drc_rules: Not Supported

Description

The named capture procedure in the procedure file cannot be loaded using the -Append_or_timing_update switch in the read_procfile command.

W33 ensures the procedure file is valid.

This DRC checks a new procedure file when it is loaded with the read_procfile or write_patterns command. A W33 violation occurs when the new procedure file contains a new scan procedure (load_unload, alternate shift, master_observe, and so on) or a new named capture procedure. W33 is only issued for new named capture procedures if the -Append_or_timing_update switch is specified with the read_procfile command.

Messaging

If the test_setup procedure file fails W33, the following message displays:

```
Cannot add new procedure P, procedure ignored (W33)
```

Where P is the procedure name.

Troubleshooting

To correct this violation, do one of the following:

- If the violation is for a named capture procedure, either change the read_procfile command option from -Append_or_timing_update to -Replace or edit the procedure file to remove the named capture procedure.
- If the violation is for a scan procedure, edit the procedure file to remove the scan procedure.
- For both cases, you can return to setup mode and load the procedure file with the add_scan_groups command.

W34

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This error occurs when saving patterns if a force event in the external mode conflicts with a pin constraint in a named capture procedure.

The error message is:

```
//Error: Event E N at time T in external mode conflicts with pin
constraint on same pin in procedure P. (W34)
```

Where E is the event type, N is the name of the event pin, T is the time at which the conflict occurs, and P is the name of the procedure that contains the conflicting pin constraint.

To resolve this issue, either the pin constraint or the named capture procedure must be changed. If this error is ignored, you will not be able to load external mode STIL or WGL patterns back into the tool because the pattern data (using the force event) will not match the pin constraint.

W35

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error (cannot be changed with the set_drc_handling command)

report_drc_rules: Supported

Description

This error occurs when a free running clock needs to be added to a cycle, and that cycle uses a timeplate that has no pulse timing for the clock.

The error message is:

```
//Error: The following occured at line N in file F
Free running clock C is not pulsed in T. (W35)
```

Where N is the line number, F is the filename, C is the clock name, and T is the timeplate name.

The W35 rule is an error, and the handling cannot be changed.

W36

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This error occurs when “offstate” statements in the timeplates used by a named capture procedure are defined for a pin that has already been defined as a clock in the dofile. The unnecessary “offstate” statements cause extra pulses in a cycle.

The error message is:

```
// Error: Offstate cannot be used with pin N already defined as a clock
(W36)
```

Where N is the name of the clock pin.

To resolve this error, remove the unnecessary “offstate” statements from the timeplates.

W37

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

Each load cycle of a named capture procedure must include a **force_pi** event. The ATPG tools require a **force_pi** event after each scan load. If there is no **force_pi** event in the load cycle of a named capture procedure, the generated test patterns may be incorrect and produce errors when read back into the ATPG tools.

The W37 violation does not override or mask a W20 violation. Therefore, if this error occurs in the first cycle and there is no force_pi event before the first clock pulse, then both a W37 and W20 violation are issued.

The W37 rule can be reduced to a warning or note only if every signal having a force event in a load_unload procedure has a pin constraint defined with a matching value.

The occurrence message is:

```
//Error: A force_pi statement does not exist in a scan load cycle in
Procedure P. (W37-1)
```

Where P is the name of the procedure.

Troubleshooting

Edit the load cycle(s) of the named capture procedure to include a force_pi event.

W38

Category: Timing

Contexts Supported: dft -scan, patterns -scan, patterns -scan_diagnosis

Default Handling: Error

report_drc_rules: Supported

Description

This rule checks to see if free running clocks that are pulsed in the external mode of a named capture procedure and are internally connected also pulse in the internal mode of the named capture procedure at the same time.

The error message is:

```
//Error: The following occured at line N in file F
Event E on a free running internally connected in at time T in external
mode does not have matching event in internal mode in procedure P. (W38)
```

Where N is the line number, F is the filename, E is the event type, T is the time at which the conflict occurs, and P is the name of the procedure that contains the conflicting pin constraint.

A PLL clock going to a sequential cell can also produce a W38 violation.

This rule defaults to error handling. If you set the rule to warning handling, then the tool attempts to auto-correct the W38 problem by copying external mode event(s) to the appropriate time in the internal mode. If the auto-correct fails (for example, a W35 violation in the internal mode), then the W38 rule's handling is changed to an error.

Other DRC Messages

DRC generates various messages to indicate the status of various processes and state elements. Some of them are discussed here.

Transparent Capture Handling Analysis

If you issue the set_capture_handling command before running DRC, an analysis message displays upon completion of DRC.

Example

```
// Begin capture handling analysis: LS=OLD,TE=OLD
//     (#C3=0 #C4=0), #user_pts=6/0
// Capture handling analysis completed: #sources=6,
//     #int_gates=34, #sinks=4, CPU_time=0 sec
```

Figure 8-37 describes each component of the messages.

Figure 8-37. Transparent Capture Handling Analysis Messages

Oscillation Limitation

The DRC simulator limits the maximum number of iterations performed to 500 while stabilizing state changes. If state elements continue to change after the maximum number of iterations is reached, the tool displays a message warning that the limit was reached. Additionally, the message reports each state element set to X to limit the iterations. State elements include latches, flip-flops, and feedback buffers.

The warning message is:

```
S forced to X in procedure P at time T due to iteration limit.
```

Where S is the state element, P is the procedure type, and T is the time.

Example

```
// latch1/Q (151) forced to X in procedure load_unload at
//     time 25 due to iteration limit.
```

RAM Summary Results and Test Capability

Upon completion of RAM rules checking, DRC displays summary result information as well as test capability information.

Example

```
// RAM Summary Results: #RAMs = 3 #TieXs = 0 #testable = 3
//     #data_hold = 3
// Test Capability: #read_only = 0 #ram_sequential = 0
//     #seq_transparent = 0
// Write stability: #unstable_control = 0 #unstable_load = 1
// Read stability:  #unstable_control = 0 #unstable_load = 3
```

Figure 8-38 describes the information displayed after running RAM rules checking.

Figure 8-38. RAM Summary Results and Test Capability Messages

Appendix A
Parameter File Format and Keywords

What is a Parameter File?

The parameter file specifies field names and values for use in certain Vector Interfaces output formats. The tool uses this file when you specify with the "write_patterns -Parameter" command.

The tool adds asection at the beginning of STIL, WGL, and Verilog output files that captures the environment data and the switches specified with the write_patterns command.

- In the STIL format, this information is added to the HEADER{} block as an annotation.
- In the WGL format, this information is added to the header block as a series of annotation statements.
- In the Verilog format, this information is added to the beginning of the Verilog testbench as a comment.

Parameter File Syntax

The parameter file is a text file containing a series of parameter statements. Each statement uses the following format:

```
KEYWORD arguments;
```

Where:

- **UPPERCASE** — Indicates a keyword and must be entered exactly as shown.
- ***Italic*** — Indicates a user-supplied argument.
- **{}** — Indicates a multi-option argument. Include only one option and do not enter the braces.
- **<u>Underline</u>** — Indicates the default setting for a multi-option argument.

The parameter file has the following rules:

- All parameter statements must end with a semicolon.
- Comments are marked with double slashes "//" at the beginning of a line. The tool ignores everything between the slashes and the end of line character.

- You can use each parameter only once per file. If you use a parameter more than once, the latter entry overrides the previous one.
- Each parameter statement must occupy a single line. You cannot embed carriage returns within a parameter statement.
- Each parameter statement line cannot exceed 2048 characters. There are no restrictions on the parameter file size.

Note

All parameters are optional, so you can include or exclude as needed. When you omit a parameter, the tool uses the default setting. Refer to each parameter description for the default behavior.

Parameter File Keywords

The following is a list of the available parameter statements:

ALL_EXCLUDE_POWER_GROUND 2683
ALL_EXCLUDE_UNUSED 2683
ALL_FIXED_CYCLES 2683
ALL_FLATTEN_TIMING. 2684
ALL_IDDQ_TESTER_CYCLE 2684
ALL_MIN_SCAN_LOAD 2685
ALL_NO_CYCLE_OPT 2685
ALL_NO_LOOP. 2686
ALL_NO_PATTERN_TYPE 2686
ALL_NONSCAN_CONSTANT 2686
ALL_ORIGINAL_INDEX 2686
ALL_TIME_RESOLUTION 2687
ALL_USE_CYCLE_LOOP 2687

CTL_ONE_PAT 2687
CTL_ONE_PROTO 2688
CTL_STIL_0. 2688
CTL_STIL_1. 2689

FTDL_DESIGNER. 2689
FTDL_MAX_PAT_BLOCK 2689
FTDL_REVISION 2689
FTDL_TEST_NAME 2690
FTDL_TEST_NUM 2690
FTDL_TEST_SUFFIX 2690
FTDL_ZMODE_MES 2690

SIM_ANNOTATE_QUIET 2691
SIM_CHAIN_ERROR 2691
SIM_CHANGE_PATH 2692
SIM_COMPARE_X 2692
SIM_DELAY_SCAN_RELEASE 2692
SIM_DIAG_FILE 2693
SIM_DISTRIBUTED_PAT 2693
SIM_EARLY_RELEASE 2694
SIM_EARLY_RELEASE_TIME 2694
SIM_FILL_BIDISCAN 2695
SIM_FORCE_Z_AT_CYCLE 2695
SIM_INCLUDE 2695
SIM_INSTANCE_NAME 2696
SIM_KEEP_PATH 2696
SIM_MASK_FILE 2696
SIM_MIN_NAME_FILE 2697
SIM_MISCOMPARE_LIMIT 2697
SIM_MODULE_INCLUDE 2698
SIM_MODULE_NAME 2698
SIM_NO_CHAINNAME_FILE 2698
SIM_NO_MEASURE_IN 2698
SIM_PARALLEL_DELAY_SHIFT_CLOCKS 2699
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE 2699
SIM_POST_SHIFT 2699
SIM_PRE_SHIFT 2700
SIM_PRECISION 2700
SIM_SERIAL_CHAIN 2700
SIM_SHIFT_DEBUG 2701
SIM_SIGNAL_SPY 2701
SIM_STATUS_MSG 2702
SIM_TIMEPLATE_COMM 2702
SIM_TOP_NAME 2702
SIM_VECTOR_COMM 2703
SIM_VECTYPE_SIGNAL 2704

STIL_2005_SCAN 2705
STIL_CHAIN_CELLS 2705
STIL_CHANNEL_CELLS 2706
STIL_FULL_CHAIN 2706
STIL_MIN_QUOTE 2706
STIL_NESTED_MACRO 2707
STIL_NO_SCANSTRUCTURE 2707
STIL_NO_SCANX 2707
STIL_NOMEASURE_CLOCK 2707
STIL_PAT_ANN 2708

STIL_PAT_CMT 2708
STIL_PAT_LAB. 2709
STIL_QUOTE_ALL. 2709
STIL_SCAN_ANN1. 2709
STIL_STRUCTURAL 2710
STIL_TI_TITLE 2712
STIL_TRIM_PULSE 2713
STIL_VECTOR_ANN 2713
STIL_VERG_ESC 2714

TITDL_CHAIN_SETTYPE 2715
TITDL_CUSTOMER 2715
TITDL_GROUP_TIMING. 2715
TITDL_KEEP_SCANOUT 2715
TITDL_LIBRARY 2716
TITDL_PARTNUM 2716
TITDL_PSEUDO_PREFIX 2716
TITDL_REVISION 2717
TITDL_SETNAME 2717
TITDL_SETTYPE 2718

WGL_ADD_LASTX 2718
WGL_ALT_BIDI 2718
WGL_ALT_VECT_ANN. 2719
WGL_EDGE_STROBE 2719
WGL_FULL_CHAIN. 2721
WGL_FULL_SCANGROUP 2722
WGL_GROUP_PIN 2722
WGL_INV_SC 2723
WGL_NO_SCANX 2723
WGL_NOMEASURE_CLOCK. 2723
WGL_ONE_ILLINOIS 2724
WGL_PATTERN_NAME 2724
WGL_TRIM_LEADING_P 2724
WGL_TRIM_PULSE 2725
WGL_VECTOR_ANN. 2725
WGL_VERG_ESC 2726
WGL_VTRAN_PADSC. 2726

A Note about the ALL_* Parameter File Keywords

All of the ALL_* parameter file keywords affect all pattern formats except for binary and ASCII.

ALL_EXCLUDE_POWER_GROUND

Usage

```
ALL_EXCLUDE_POWER_GROUND {AUTO | 1 | 0};
```

Description

This keyword excludes any port with a “function” value of “power” or “ground” from the written pattern.

By default, the keyword excludes such ports in tester formats, such as WGL and STIL, and does not exclude such ports for Verilog or VHDL. Setting this keyword to 1 excludes power and ground ports from all formats (except binary and ASCII); setting this keyword to 0 prevents exclusion of power and ground ports.

Note that binary and ASCII patterns are not affected by this keyword.

ALL_EXCLUDE_UNUSED

Usage

```
ALL_EXCLUDE_UNUSED {0 | 1};
```

Description

When set to 1 or true, this keyword removes the following pins from patterns:

- All output pins that are masked.
- All input pins that are constrained to CX and are not used in any test procedures.
- All bidi pins that are constrained to CX or CZ and are either not used in any test procedures or are only forced to Z and masked in the test procedures.

Note that this keyword does not remove pins from a multi-bit bus unless it can remove the entire bus.

ALL_FIXED_CYCLES

Usage

```
ALL_FIXED_CYCLES {2 | 1 | 0};
```

Description

Issuing a 2 with this keyword causes the tool to compute the largest number of cycles between scan loads by looking at all patterns in the pattern list, not just those being saved. Issuing a 1

with this keyword specifies that the patterns will contain an equal number of non-scan cycles between each scan chain loading. This non-scan cycle is applied before the first shift cycle. The default is 0.

This will add “Shift” cycles before the regular apply shift of the load_unload procedure. It adds these additional cycles before the scan chain load to make all patterns have the same total number of cycles per pattern, taking into account the differences in the number of capture cycles. The number of capture cycles can still vary depending on what ATPG needed or was directed with the Named Capture Procedures. However, no additional “Capture” cycles are added. The new cycles will use the timeplate that the “Shift” procedure uses.

ALL_FLATTEN_TIMING

Usage

```
ALL_FLATTEN_TIMING {1 | 0};
```

Description

Issuing a 1 with this keyword causes all outputs to compute existing timing equations and use only the resulting numeric values in the output files. The default of 0 will result in any output format which can support equation-based timing using the equations as they are specified.

This keyword is intended for having pattern output files not contain equation-based timing that has been added to the procedure file and preserved in the tool. Test data languages such as Verilog, WGL, and STIL have the ability to express time values in the timing blocks, as numerical values or as equations based on variables. Using equation-based timing allows one value to be specified for a global attribute, such as the test cycle period, while other values are derived from this using equations.

ALL_IDDQ_TESTER_CYCLE

Usage

```
ALL_IDDQ_TESTER_CYCLE {1 | 0};
```

Description

Issuing a 1 with this keyword forces the IDDQ file to contain tester cycle counts and serial simulation times even if the pattern file being saved is a parallel load pattern file. Not using the keyword, or using it with a 0 argument invokes the default behavior where the IDDQ file uses tester cycle counts for serial pattern formats, and vector counts and parallel load times for parallel formats.

In the example, saving a parallel load pattern file for a particular design and using the -iddq_file, the IDDQ file has the following content:

```
/* Pattern # Cycle # Time */
0 71 7100
1 76 7600
...
```

Saving the same parallel load pattern file and using -iddq_file, but also using ALL_IDDQ_TESTER_CYCLE 1; produces an IDDQ file with the following content:

```
/* Pattern # Cycle # Time */
0 195 19500
1 324 32400
...
```

The cycle counts and times given now correspond to serially loaded tester cycle counts and simulation time stamps

ALL_MIN_SCAN_LOAD

Usage

```
ALL_MIN_SCAN_LOAD {1 | 0};
```

Description

By default, all pattern outputs contain scan-in and scan out data for each scan load. Setting this keyword to 1 allows a scan load to specify only scan-in or scan out when appropriate (first scan in). The default is 0.

ALL_NO_CYCLE_OPT

Usage

```
ALL_NO_CYCLE_OPT {1 | 0};
```

Description

Issuing a 1 with this keyword specifies for all outputs to turn off cycle optimizations in non-scan areas of patterns and leave all cycles in the patterns. Issuing a 0 with this keyword (or omitting this keyword as it is the default), will allow the following optimization in cycles that are part of the Capture or Clock procedures: If any two identical cycles are generated and do not measure any outputs, the duplicate cycles will be removed.

ALL_NO_LOOP

Usage

```
ALL_NO_LOOP {1 | 0};
```

Description

Disables the use of “loops” or “repeat” statements to represent sub_procedures. The default is off (0). For more information, refer to “Disabling Sub_procedure Looping” in the *Tessent Shell User’s Manual*.

ALL_NO_PATTERN_TYPE

Usage

```
ALL_NO_PATTERN_TYPE {1 | 0};
```

Description

If set to 1, this keyword eliminates the pattern type annotation. This keyword applies to all Vector Interface formats. The default is 0.

Caution

This keyword should only be used if the pattern type annotations are causing problems. If you set this keyword to 1, you will not be able to load the generated patterns back into the DFT tools.

ALL_NONSCAN_CONSTANT

Usage

```
ALL_NONSCAN_CONSTANT {1 | 0};
```

Description

Setting this keyword to 1 forces all non-scan pins, excluding control and clock pins, to the constant value of the pad state for all scan patterns. This keyword may be useful, as an alternative to procedure force statements, for testers that require constant pin values. The default is 0.

ALL_ORIGINAL_INDEX

Usage

```
ALL_ORIGINAL_INDEX {1 | 0};
```

Description

When set to 1, saved pattern files use the original pattern index (from before filtering) for the annotations and comments prior to each pattern. When set to 0 (the default), the pattern numbering in the comments and annotations is numbered in order starting at 0 from the first

pattern as if the filtered patterns are a new pattern set. Note that this affects all types of pattern outputs other than Verilog because the Verilog testbench uses a built-in pattern count mechanism.

ALL_TIME_RESOLUTION

Usage

```
ALL_TIME_RESOLUTION integer;
```

Description

This keyword is used to specify the number of significant bits for floating point time numbers. The maximum is 4, and the default is 3.

ALL_USE_CYCLE_LOOP

Usage

```
ALL_USE_CYCLE_LOOP {1 | 0};
```

Description

Uses updated terminology in the generated pattern files to prevent confusion with Tessent Diagnosis terms. The previous “cycle” and “loop” terms in the pattern files are replaced with “simulation cycle” or “vector” and “tester cycle”. Set this keyword to 1 to revert back to the “cycle” and “loop” terminology.

CTL_ONE_PAT

Usage

```
CTL_ONE_PAT {1 | 0};
```

Description

This keyword, if set to 1, causes all pattern data to be grouped into a single STIL pattern block, instead of one pattern block per ATPG pattern. The default is 0.

CTL_ONE_PROTO

Usage

```
CTL_ONE_PROTO {1 | 0};
```

Description

This keyword, if set to 1, disallows protocol-level macros. The default of 0 allows protocol-level macros.

Note

In addition to the three CTL parameter file keywords listed above, all parameter file keywords for STIL output also work with CTL output. The tool-generated CTL output can be customized using the above three parameters to be a more structural STIL output.

CTL_STIL_0

Usage

```
CTL_STIL_0 {1 | 0};
```

Description

This keyword, if set to 1, causes the output to be backward compatible with the IEEE 1450.6 standard. The Environment block will be dropped and the STIL header will not mention the CTL extension. The default is 0.

Note

CTL (IEEE 1450.6) is an extension of STIL that creates a standard format to describe IP core and SOC test information. IEEE 1450.6 is a Draft Standard. As of the date of this release, the CTL output is in compliance with the “Complete Draft 0.3” version of this standard. As changes and upgrades are made to the standard, the output will be upgraded.

CTL_STIL_1

Usage

```
CTL_STIL_1 {1 | 0};
```

Description

This keyword, if set to 1, causes the output to eliminate all 1450.6 syntax, but leave the IEEE 1450.1 syntax. The default is 0, which produces the full 1450.6 and 1450.1 syntax used in the CTL output.

The CTL_STIL_0 parameter file keyword can be used in the CTL output to eliminate the 1450.1 features along with all of the 1450.6 syntax.

For more information, refer to the "write_patterns -CTL" command description in the *Tessent Shell Reference Manual*.

FTDL_DESIGNER

Usage

```
FTDL_DESIGNER string;
```

Description

Specifies an informational string to be used as the designer name in the DESIGNER statement at the top of the FTDL file. If not specified, the default is "ATPG".

FTDL_MAX_PAT_BLOCK

Usage

```
FTDL_MAX_PAT_BLOCK integer;
```

Description

Specifies the maximum number of vector statements allowed in a single TEST block, including all shifted values and repeated vectors. If not specified, the default is 5 million.

FTDL_REVISION

Usage

```
FTDL_REVISION string;
```

Description

Specifies an informational string to be used as the revision string in the REVISION statement at the top of the FTDL file. If not specified, the default is "0001".

FTDL_TEST_NAME

Usage

```
FTDL_TEST_NAME string;
```

Description

Specifies the root name to be used for the test block(s) within the FTDL file. If not specified, the default root name is “FN”. The root name is combined with a test number to create each test block name.

FTDL_TEST_NUM

Usage

```
FTDL_TEST_NUM integer;
```

Description

Specifies the starting test block number, to be appended to the root test block name. For each additional test block needed in the FTDL file, the test number is incremented. If not specified, the starting test block number is 1.

FTDL_TEST_SUFFIX

Usage

```
FTDL_TEST_SUFFIX string;
```

Description

Specifies a string to add as a suffix to the test block name. The total test block name is constructed from FTDL_TEST_NAME, FTDL_TEST_NUM, and optionally, FTDL_TEST_SUFFIX. The required argument *string* is the suffix that will be added after the test block number. The default is to have no suffix.

FTDL_ZMODE_MES

Usage

```
FTDL_ZMODE_MES {1 | 0};
```

Description

Controls whether the FTDL file allows for Z values to be measured on output pins and output sides of bidi pins. If set to 0, then the “ZMODE = NOMES” statement will be placed in all TEST blocks within the FTDL file, and no Z values will be used on output pins. If set to 1, then the “ZMODE = MES” statement are placed in all TEST blocks, and Z values are allowed on output pins. The default is 0, NOMES.

Keywords FTDL_TEST_NAME and FTDL_TEST_NUM are used together to create the name(s) of the TEST block(s) within the FTDL file.

Keyword FTDL_MAX_VEC_BLOCK is used to set the maximum number of vector statements allowed in a single TEST block within the FTLD file.

Keywords FTDL_REVISION and FTDL_DESIGNER simply supply informational strings to be used within the FTDL file.

Keyword FTDL_ZMODE_MES controls whether ZMODE = MES, or ZMODE = NOMES, and is used in the FTDL file.

The following is an example parameter file for FTDL.

```
FTDL_TEST_NAME "MyTest";
FTDL_REVISION "001.1";
FTDL_DESIGNER "Smith";
FTDL_ZMODE_MES 1;
```

SIM_ANNOTATE_QUIET

Usage

```
SIM_ANNOTATE_QUIET {1 | 0};
```

Description

When set to 1, this keyword suppresses the Verilog testbench from displaying the annotations during simulation. The default behavior is to display the annotations.

SIM_CHAIN_ERROR

Usage

```
SIM_CHAIN_ERROR {1 | 0};
```

Description

Setting this parameter to 1 adds a register/signal per scan chain to the Verilog testbench. This signal may be monitored during simulation and goes to 1 when a mismatch is detected on the scan chain. The name of the register/signal is mgcdft_*chainname*_error where *chainname* is the chain name specified on the add_scan_chains command line. The default is 0.

SIM_CHANGE_PATH

Usage

```
SIM_CHANGE_PATH {1 | 0};
```

Description

Setting this parameter to 1 inserts code into the test bench that allows you to change the path to the files during simulation. Use the VERILOG PlusArgs keyword “NEWPATH=” to pass a new file path to test bench. For example:

```
vsim top_CngP_v_ctl +NEWPATH=./results/ -c -do "run -all"
```

Note

SIM_CHANGE_PATH and SIM_KEEP_PATH are mutually exclusive. SIM_CHANGE_PATH is ignored if SIM_KEEP_PATH is present.

SIM_COMPARE_X

Usage

```
SIM_COMPARE_X {1 | 0};
```

Description

When set to 1, this parameter adds code to the testbench to not mask X states, but to compare for differences. This functionality is available in both Serial and Parallel testbenches. Issuing a 0, the default behavior, causes the testbench to not include these capabilities.

Note

While issuing a mismatch if we expect X is not required for ATPG, it is necessary for LibComp verification to identify cases where the translation is pessimistic; otherwise, modeling a state such as a Tie-X will not look like a problem since you won’t get mismatches.

SIM_DELAY_SCAN_RELEASE

Usage

```
SIM_DELAY_SCAN_RELEASE integer;
```

Description

This keyword and positive integer pair delays the scan release by the specified number. The unit of the specified *integer* is the same as the timescale units. If there are multiple chains, each release is delayed by the specified *integer*.

SIM_DIAG_FILE

Usage

```
SIM_DIAG_FILE {2 | 1 | 0};
```

Description

Controls the generation of the failure file in the test bench. The failure file name is generated by appending the test bench file name with ".fail". For example, a test bench file named "testbench.v" would generate a failure file named "testbench.v.fail".

For uncompressed ATPG, this keyword works for both serial and parallel Verilog testbenches. For compressed ATPG, the keyword only works for the serial Verilog testbenches and has no impact on the parallel testbench.

For "patterns -scan" context, the default setting is 1. For all other contexts, the default is 0. The options for this parameter are as follows:

- **0** — No Verilog code for generating the failure file is included in the test bench. This setting maintains backward compatibility with previous test bench versions.
- **1** — Verilog code for generating the failure file is included in the test bench, but the file is not automatically generated. To activate generation of the failure file, you must edit the test bench, setting the "_write_DIAG_file" parameter to 1.
- **2** — Verilog code for generating the failure file is included and activated in the test bench. You can explicitly disable failure file generation by setting the "_write_DIAG_file" parameter to 0 in the test bench.

SIM_DISTRIBUTED_PAT

Usage

```
SIM_DISTRIBUTED_PAT {1 | 0};
```

Description

Turns off the enhanced Verilog testbench that allows you to choose which patterns to simulate. When set to 0, all saved patterns are simulated. When set to 1 (the default), this statement allows you to choose which patterns to simulate.

You can then use the STARTPAT and ENDPAT PlusArgs keywords to specify the starting and ending patterns. For example, the following runs the simulation using pattern files 2 through 10:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=2 +ENDPAT=10
```

If you specify only the ENDPAT keyword, the simulation begins with the first pattern or file and end at the specified pattern or file. For example, the following runs the simulation for the first four patterns:

```
vsim simple_testbench_v_ctl -c -do vsim.do +ENDPAT=4
```

If you specify only the STARTPAT keyword, the simulation starts with the specified pattern or file and end at the last pattern or file. For example, the following runs the simulation for pattern files 3 through 20, where 20 is the last pattern file:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=3
```

To simulate a single pattern, specify the same value both the start and end keywords. For example, the following runs the simulation for only pattern 8:

```
vsim simple_testbench_v_ctl -c -do vsim.do +STARTPAT=8 +ENDPAT=8
```

SIM_EARLY_RELEASE

Usage

```
SIM_EARLY_RELEASE {1 | 0};
```

Description

Designates the Verilog testbench to put the “release” statement immediately after the trailing edge of the first clock pulse in the shift procedure. Issuing a 0, the default behavior, will place the “release” statement at the end of the cycle.

SIM_EARLY_RELEASE_TIME

Usage

```
SIM_EARLY_RELEASE_TIME integer;
```

Description

Specifies exactly what time you want the Verilog testbench “release” statement to happen using the time scale specified in the procedure file. The integer value is a time value that is required to be in the same time scale as that used in the timeplate for the shift procedure. The time specified must be later than the time of the force_sci event, but not greater than the time of the end of the cycle that the force_sci event occurs in. The time is relative to the start of the cycle.

This keyword does not replace the SIM_EARLY_RELEASE keyword, but can be used as an alternative. If both the SIM_EARLY_RELEASE and the SIM_EARLY_RELEASE_TIME keywords are specified in the parameter file, then the SIM_EARLY_RELEASE_TIME keyword takes precedence. If neither keyword is used, then the default behavior of putting the “release” statement at the end of the cycle is still used.

Note

The SIM_EARLY_RELEASE and SIM_EARLY_RELEASE_TIME statements are helpful when complex scan cells that use multiple clocks need the release to happen at a certain time prior to the end of the shift procedure, in order for the parallel load testbench to load the correct values. That is, if the forced value is held too long, a second clock pulse will latch in the wrong value.

SIM_FILL_BIDISCAN

Usage

```
SIM_FILL_BIDISCAN {1 | 0};
```

Description

By default, the bidirectional scan_input pins are set to “X’ during parallel scan loading in the Verilog test bench. Setting this keyword to 1 sets the bidirectional scan_input pins to the scan-in value of the first cell.

SIM_FORCE_Z_AT_CYCLE

Usage

```
SIM_FORCE_Z_AT_CYCLE {1 | 0};
```

Description

Controls when a bidi pin switches to a Z state in the Verilog testbench. The default behavior is that when a bidi pin is going to change to a Z value during a given simulation vector, then the Z is forced at time 0 for that vector. However, when you set the keyword value to 0, the Z value applies to the bidi pin at the force time specified in the timeplate rather than at time 0. Note that if the force time happens after the internal driver for the bidi pin turns on, bus contention is possible.

SIM_INCLUDE

Usage

```
SIM_INCLUDE filename;
```

Description

Adds an include statement into the Verilog testbench after the comments and before the first module. The contents of the filename specified must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The Default behavior is to not add any include statements.

SIM_INSTANCE_NAME

Usage

```
SIM_INSTANCE_NAME string;
```

Description

Specifies the instance name for your design, to be used within the testbench.

For example, if your current design is simple_chain2 and you want to specify the instance name presy, this is the entry in the parameter file:

```
SIM_INSTANCE_NAME presy ;
```

And this is the test bench:

```
//   top_module_name   = simple_chain2_pat_par_v_ctl
module simple_chain2_pat_par_v_ctl;


simple_chain2 presy (.reset(reset), .clk(clk), .in1(in1),
     .scan_in1(scan_in1), .scan_en(scan_en), .out0(out0),
     .out1(out1), .out2(out2), .out3(out3), .all0(all0),
     .all1(all1), .w1out(w1out), .scan_out1(scan_out1));
```

SIM_KEEP_PATH

Usage

```
SIM_KEEP_PATH {1 | 0};
```

Description

Causes the Verilog testbench to use the full pathname to specify the pattern data file. Issuing a 0, the default behavior, causes the Verilog testbench to specify the pattern data file without any path information.

Note

SIM_CHANGE_PATH and SIM_KEEP_PATH are mutually exclusive. SIM_CHANGE_PATH is ignored when SIM_KEEP_PATH is present.

SIM_MASK_FILE

Usage

```
SIM_MASK_FILE {2 | 1 | 0};
```

Description

Controls the generation of the mask file, which captures simulation mismatches, in the test bench. The mask file name is generated by appending the test bench file name with “.mask”.

For example, a test bench file named “testbench.v” would generate a mask file named “testbench.v.mask”.

For “patterns -scan” context, the default setting is 1. For all other contexts, the default is 0.

- **0** — No Verilog code for generating the mask file is included in the test bench. This setting maintains backward compatibility with previous test bench versions.
- **1** — Verilog code for generating the mask file is included in the test bench, but the file is not automatically generated. To activate generation of the mask file, you must edit the test bench, setting the “_write_MASK_file” parameter to 1.
- **2** — Verilog code for generating the mask file is included and activated in the test bench. You can explicitly disable mask file generation by setting the “_write_MASK_file” parameter to 0 in the test bench.

Note

The simulation mismatches for the ChainTest patterns are recorded in the SIM_MASK_FILE, but they are commented out.

SIM_MIN_NAME_FILE

Usage

```
SIM_MIN_NAME_FILE {1 | 0};
```

Description

Controls the number of *.name* output files generated when saving patterns as a Verilog test bench. Setting this keyword to 0 causes the tool to generate a separate set of *.name* files for each test bench file. When using the -maxloads switch for large designs with many scan chains and patterns, this generates a very large number of files. Setting this keyword to 1 reduces the number of output files by generating only one set of *.name* files for a design. The same *.name* files can be used for all generated Verilog test benches for that design.

This parameter file keyword is on by default when you use the write_patterns command with the -maxloads switch. It is off by default otherwise.

SIM_MISCOMPARE_LIMIT

Usage

```
SIM_MISCOMPARE_LIMIT integer;
```

Description

Sets a limit in the Verilog testbench for the number of miscompares to allow before the simulation terminates. By default, the testbench runs the simulation until all saved patterns are simulated, regardless of how many miscompares happen. This keyword causes the simulation to abort upon reaching the miscompare limit.

SIM_MODULE_INCLUDE

Usage

```
SIM_MODULE_INCLUDE filename;
```

Description

Similar to the SIM_INCLUDE parameter, except it adds the include statement within the module definition for the testbench instead of after the comments and before the first module. The *filename* can be any valid file path. The contents of the specified file must contain legal Verilog statements and syntax. This statement is used only in the Verilog outputs. The default behavior is to not add any include statements.

SIM_MODULE_NAME

Usage

```
SIM_MODULE_NAME string;
```

Description

Specifies a string that overrides the module name of the DUT in the Verilog simulation testbench.

SIM_NO_CHAINNAME_FILE

Usage

```
SIM_NO_CHAINNAME_FILE {1 | 0};
```

Description

Setting this keyword to 1 prevents the tool from generating the *filename*.chain.name file. This file is generated by default.

SIM_NO_MEASURE_IN

Usage

```
SIM_NO_MEASURE_IN {1 | 0};
```

Description

In the Verilog testbench, when a value is being forced on a bidirectional pin (as in input mode), the testbench also tries to measure the same value on the bidirectional pin when a measure happens. Setting this keyword to 1 causes the testbench to not measure input values on the output side of bidirectional pins. The default is 0.

When you issue a “write_patterns -Verilog -mode LSI” command, the tool automatically sets SIM_NO_MEASURE_IN to 1.

SIM_PARALLEL_DELAY_SHIFT_CLOCKS

Usage

```
SIM_PARALLEL_DELAY_SHIFT_CLOCKS {integer};
```

Description

Delays the leading edge of all shift clocks in all shift procedures by the specified number of timescale units. Any events occurring after the leading edge of the first shift clock is also delayed to maintain the event order. The specified number must be a positive integer. If the specified delay causes any event to occur beyond the period of the timeplate, the tool ignores this keyword.

This keyword is valid only for Verilog parallel test bench generation.

SIM_PARALLEL_EARLY_FORCE_AND_MEASURE

Usage

```
SIM_PARALLEL_EARLY_FORCE_AND_MEASURE {1 | 0 | true | false};
```

Description

Setting this keyword to 1 or true moves the measure_sco event to time 0 and the force_sci event to time 1. By default (0 or false), the measure_sco event occurs at the measure_po time specified in the timeplate for the shift procedure. The force_sci event then occurs one timescale unit later, regardless of the force_pi time specified in the timeplate.

This keyword is valid only for Verilog parallel test bench generation.

SIM_POST_SHIFT

Usage

```
SIM_POST_SHIFT integer;
```

Description

Specifies the absolute number of independent post shifts to use during simulation. That is, if the load_unload procedure already has one post shift, setting this keyword to “2” causes two post shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of post shifts to the total number of shifts minus one.

You can use this keyword to generate a parallel simulation testbench that applies a parallel load followed by N number of serial shift cycles. When used in the “patterns -scan” context, SIM_POST_SHIFT serial shift cycles are applied by forcing the scan-in values onto the scan pin of the design. You can also use this keyword to generate an EDT parallel simulation

testbench; in this case, the serial shifts are performed on the EDT internal scan chains and the testbench forces values onto the scan ports driven by the decompressor.

However, you cannot use this keyword to generate an EDT parallel simulation testbench in which the serial inputs are shifted in through the EDT channels because serial simulation through the decompressor and other EDT logic is not supported by parallel testbench.

SIM_PRE_SHIFT

Usage

```
SIM_PRE_SHIFT integer;
```

Description

Specifies the absolute number of independent pre shifts to the load_unload procedure (prior to main shift) to use during simulation. That is, if the load_unload procedure already has one pre shift, setting this keyword to “2” causes two pre shifts to be used rather than three.

If you specify a negative number or zero, the tool ignores this keyword. If you specify a number larger than the total number of shifts, the tool sets the number of pre shifts to the total number of shifts minus one.

SIM_PRECISION

Usage

```
SIM_PRECISION string;
```

Description

Specifies the timescale precision. The string is a number and a unit, for example:

```
SIM_PRECISION 1ns;
SIM_PRECISION 10ms;
```

The units supported are ms, us, ns, ps, and fs.

The tool checks this value to verify that it is less than or equal to the Timescale unit from the procedure file and outputs this value in the test benches.

SIM_SERIAL_CHAIN

Usage

```
SIM_SERIAL_CHAIN {1 | 0};
```

Description

Creates a test bench that applies chain test patterns in serial mode and all other patterns in parallel mode (when set to 1). The keyword is set to 0 by default.

SIM_SHIFT_DEBUG

Usage

```
SIM_SHIFT_DEBUG integer;
```

Description

Specifies for the serial Verilog testbench to use special test patterns that contain a parallel observe of the entire scan chain for every shift, while serially shifting the patterns.

For debugging purposes only, this format makes it easier to pinpoint precisely which scan cell in a chain is causing a problem. Issuing a 0, which is the default behavior, causes the serial Verilog testbench to use test patterns that do not contain a parallel observe of the entire scan chain for every shift.

An integer value of 1 causes the testbench to shift serially and do a parallel compare for all shifts. An integer value of 2 or greater causes the testbench to shift serially and do parallel compares on the first 2 or greater shifts. This is done to reduce the amount of data in the pattern files.

The Verilog patterns you create using the keyword SIM_SHIFT_DEBUG typically take more time for the tool to create than other kinds of patterns and contain large amounts of data (so file sizes tend to be huge). Also, because they simulate serially, the patterns take more time to simulate. Be sure these are acceptable trade-offs before trying to use these patterns.

The tool ignores the SIM_SHIFT_DEBUG keyword unless you include the "write_patterns -Serial" switch. Also, if you do not include the -End or "-pattern_sets chain" options, the tool defaults to "-End 0", saving chain test patterns and the first scan pattern. To save chain test patterns and a different set of scan patterns, use the -Begin and/or -End switches to specify the range of scan test patterns to save. To save only chain test patterns, include the "-pattern_sets chain" option.

Note that this functionality is available only for patterns created in the "patterns -scan" context and EDT bypass patterns (uncompressed).

SIM_SIGNAL_SPY

Usage

```
SIM_SIGNAL_SPY {1 | 0};
```

Description

Specifies to use the SignalSpy functions $signal_force, $signal_release, and $init_signal_spy (for observing the scan outs) to do a parallel load of the scan chain (when set to 1). There are three integer variables in the test bench that you can change from "0" to "1" to get verbose messaging from the SignalSpy functions. The variables are _ssi_verbose for $init_signal_spy,

_ssf_verbose for $signal_force, and _ssr_verbose for $signal_release. The default is 0, which specifies to use the standard Verilog force and release to do a parallel load of the scan chain.

These functions work only with the ModelSim Verilog simulator. The SignalSpyPathSeparator in the modelsim.ini file must not be set or be set to the default of "/".

SIM_STATUS_MSG

Usage

```
SIM_STATUS_MSG integer;
```

Description

Specifies the frequency of status messages output from the Verilog simulation testbenches. This keyword accepts an integer that specifies the interval (number of patterns) to wait before outputing a message. The simulation message is "Simulated *n* patterns."

SIM_TIMEPLATE_COMM

Usage

```
SIM_TIMEPLATE_COMM {1 | 0};
```

Description

Turns on timeplate comments in the vector file. The default is off (0). When you set this parameter to 1, the tool adds comments for the timeplate to the vector file. The format of the timeplate comment is:

```
// Timeplate: <timeplate_name>
```

SIM_TOP_NAME

Usage

```
SIM_TOP_NAME string;
```

Description

Specifies a new top level module name for the Verilog testbench. The default behavior is to generate a default top level module name based on the design name and the name of the pattern file.

SIM_VECTOR_COMM

Usage

```
SIM_VECTOR_COMM {1 | 0};
```

Description

Set this keyword to 1 to turn on vector type comments in the vector file, the default is off. The format of the vector type comment is:

```
// Vector type: vector_type_string
```

Where *vector_type_string* is one of the vector types listed in Table A-1.

Table A-1. Vector Types

Vector Type	Definition
TEST_SETUP	Vector from test_setup procedure.
TEST_END	Vector from test_end procedure.
SEQ_TRANSPARENT	Vector from seq_transparent procedure.
CLOCK_PROCEDURE	Vector from clock procedure.
CLOCK_PROCEDURE_RESTORE	Vector from clock procedure that uses restore_bidi event.
LOAD_UNLOAD	Vector from load_unload procedure.
SHIFT	Vector from shift procedure.
SINGLE_SHIFT	Single shift vector applied in load_unload procedure.
SHIFT_EXTRA	Vector from multicycle shift procedure that does not contain the force_sci and measure_sco events.
SHADOW_CONTROL	Vector from shadow_control procedure.
MASTER_OBSERVE	Vector from master_observe procedure.
SHADOW_OBSERVE	Vector from shadow_observe procedure.
SKEW_LOAD	Vector from skew_load procedure.
PRIMARY[1]	Vector from a non-scan procedure such as capture, named capture, clock_sequential, or ram_sequential.
PRIMARY_MEASURE[1]	Vector from a non-scan procedure that measures primary outputs.
PRIMARY_CLOCKPO[1]	Vector from a clock_po procedure.

1. In addition to the above, the “PRIMARY” vector types can have “_IDDQ” appended to the end to indicate a iddq measure point.

SIM_VECTYPE_SIGNAL

Usage

```
SIM_VECTYPE_SIGNAL {1 | 0 | 2};
```

Description

When set to 0, specifies that all vector types in the currently active procedure be represented in the pattern file as a signal, and that these signals be “1” when that vector type is active. This allows you to specify the patterns for tracing and viewing so you can see what type of vector is currently active at any specified moment within a simulation. For an example of using this keyword, refer to “Analyzing the Simulation Data” in the *Tessent Scan and ATPG User's Manual*.

When set to 1, this keyword adds an encoded vector type to the Verilog or VHDL testbench in the bus MGCDFT_VECTYP[0-3]. Table A-2, lists the signal or variable names added to the testbench that map to the bus encoding. Setting the keyword to 0 causes the testbench to not include these capabilities.

Table A-2. Translation Table

MGCDFT_VECTYP[0-3] Encoding	Signal/Variable
0001	mgcdft_test_setup
0010	mgcdft_load_unload
0011	mgcdft_shift
0100	mgcdft_single_shift
0101	mgcdft_shift_extra
0110	mgcdft_shadow_control
0111	mgcdft_master_observe
1000	mgcdft_shadow_observe
1001	mgcdft_skew_load
1010	mgcdft_seq_transparent
1011	mgcdft_launch_capture
1100	mgcdft_shift
1101	mgcdft_clock_proc
1110	mgcdft_clock_proc_restore
1111	mgcdft_test_end
0000	mgcdft_unknown

Note

Adding signals indicating vector type to the Verilog test bench enables you to see what type of vector is currently active at any specified moment within a simulation and can simplify simulation mismatch debugging.

Setting this keyword to 2 (the default), which is available only with a Verilog test bench, also adds the encoded vector types (default behavior). Additionally, the test bench will have the following _procedure_strings, which can be added to your WAVE display to show the currently active Procedure Type.

CLOCK_PROC	SEQ_TRANSPARENT	SINGLE_SHIFT
CLOCK_PROC_RES	SHADOW_CONTROL	SKEW_LOAD
LAUNCH_CAPTURE	SHADOW_OBSERVE	TEST_SETUP
LOAD	SHIFT	TEST_END
MASTER_OBSERVE	SHIFT_EXTRA	UNKNOWN

STIL_2005_SCAN

Usage

```
STIL_2005_SCAN {1 | 0};
```

Description

Issuing a 1 with this keyword is similar to the STIL_STRUCTURAL keyword except it also uses the IEEE 1450.1 ScanStructures block. The default is 0, which produces the existing IEEE 1450.0 ScanStructures block.

With the STIL_2005_SCAN keyword, the CTL output uses the 1450.1 ScanStructures block by default. You can use the CTL_STIL_0 parameter file keyword in the CTL output to eliminate the new 1450.1 features along with all of the 1450.6 syntax. The parameter file keyword CTL_STIL_1 can be used to eliminate all 1450.6 syntax but leave the 1450.1 syntax.

Note that the ScanStructures in your STIL structure must match the compressed structure.

In order to use this output, you must use the “write_patterns -STIL1999” command.

STIL_CHAIN_CELLS

Usage

```
STIL_CHAIN_CELLS {1 | 0};
```

Description

Setting this keyword to 0 removes the ScanCells list from the ScanChain definition in the ScanStructures block when saving non-compressed scan patterns that use scan chains. The following example shows a ScanStructures block without the ScanCells list:

```
ScanStructures {
```

```
      ScanChain chain1 {
         ScanLength 100;
         ScanInversion 0;
         ScanIn "sci";
         ScanOut "sco";
      }
   }
```

By default (1), the scan cell names are listed following the ScanLength statement.

For compressed patterns, see the STIL_CHANNEL_CELLS keyword.

STIL_CHANNEL_CELLS

Usage

```
STIL_CHANNEL_CELLS {1 | 0};
```

Description

Enables the listing of dummy scan cell names in the ScanStructures block when saving compressed patterns that use EDT channels. By default (0), the dummy scan cell names are not listed.

For non-compressed patterns, see the STIL_CHAIN_CELLS keyword.

STIL_FULL_CHAIN

Usage

```
STIL_FULL_CHAIN {1 | 0};
```

Description

Causes the tool to shorten the list of scan cells by the number of pre shifts and post shifts. By default (1), all cells in the scan chain are listed in the STIL file regardless the number of pre shifts and post shifts.

STIL_MIN_QUOTE

Usage

```
STIL_MIN_QUOTE {1 | 0};
```

Description

Issuing a 1 with this keyword is for STIL2005 and CTL outputs, and causes the tool to write the STIL filebusing minimum quoting of signal names that use reserved characters. Further, the existing STIL1999 output can be changed to use the same minimum quoting rules. The default is 0 for the STIL1999 output.

STIL_NESTED_MACRO

Usage

```
STIL_NESTED_MACRO {1 | 0};
```

Description

Setting this keyword to 1 when saving STRUCTURAL_STIL, STIL2005, or CTL patterns causes sub procedures to be interpreted as Macros. As Macros, they are nested inside the calling Macros. The tool uses loop statements, if necessary, to repeat the Macro. This keyword is set to 0 by default.

STIL_NO_SCANSTRUCTURE

Usage

```
STIL_NO_SCANSTRUCTURE {1 | 0};
```

Description

Setting this keyword to 1 removes the entire ScanStructures block from the STIL output. For design flows where the ScanStructures block is not required, removing the block significantly reduces the size of the STIL file. The ScanStructures block is included by default (0).

STIL_NO_SCANX

Usage

```
STIL_NO_SCANX {1 | 0};
```

Description

If set to 1, this keyword replaces all X scan load values in the STIL pattern output with 0 scan load values. The default is 0.

STIL_NOMEASURE_CLOCK

Usage

```
STIL_NOMEASURE_CLOCK {1 | 0};
```

Description

Issuing a 1 with this keyword drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. To deactivate, issue a 0 (the default) with this keyword.

Examples follow:

```
Parameter file : default
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
            01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
                                    STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '200ns' U/D; '300ns' D } }
```

STIL_PAT_ANN

Usage

STIL_PAT_ANN *string %special_keyword string2*;

Description

Specifies the annotation string that the tool outputs to the STIL pattern file just prior to each scan pattern.

```
annotation "string %keywords string"
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Parameter file entry:

```
STIL_PAT_ANN   "Pattern Number:%pat_num
(%pat_num_plus1), Cycle Number:%cycle_num"
```

Resultant pattern file statement:

```
Ann {* "Pattern Number:0 (1), Cycle Number:0" *}
```

STIL_PAT_CMT

Usage

STIL_PAT_CMT {1 | 0};

Description

Turns off (0) the pattern comment that precedes each ATPG pattern. The default is 1.

STIL_PAT_LAB

Usage

```
STIL_PAT_LAB {1 | 0};
```

Description

Turns off (0) the pattern label that precedes each ATPG pattern. The default is 1.

STIL_QUOTE_ALL

Usage

```
STIL_QUOTE_ALL {1 | 0};
```

Description

Causes all user identifiers to be enclosed in double quotes in the STIL output file. By default (0), user identifiers are not quoted.

If the STIL_QUOTE_ALL and STIL_MIN_QUOTE keywords are both active (1), the tool ignores the STIL_MIN_QUOTE keyword.

STIL_SCAN_ANN1

Usage

STIL_SCAN_ANN1 *string %special_keyword string2*;

Description

Specifies the string the tool outputs to the STIL pattern just prior to the first scan pattern. There is only one STIL_SCAN_ANN1 statement written for each pattern file generated.

```
annotation “string %keywords string”
```

The following example shows lines in a parameter file and the resultant text generated in the output pattern file.

Parameter file entry:

```
STIL_SCAN_ANN1 "Scan Chain:%chain, length:%length,
SI:%scan_in, SO:%scan_out"
```

Resultant pattern file statement:

```
Ann {*
"Scan Chain:chain1, length:2, SI:scan_in1, SO:Q" *}
```

STIL_STRUCTURAL

Usage

```
STIL_STRUCTURAL {1 | 0};
```

Description

Causes the STIL file to be written using a more structural form. This keyword set to 1 results in a STIL output form that uses some of the basic hierarchical approaches of CTL output, but not any of the IEEE 1450.6 CTL syntax extensions. As with CTL output, the structural STIL output creates STIL Macro definitions for all used procedures, and a STIL Procedure definition for the test_setup procedure, if present. The start of each pattern still consists of an annotation statement and a pattern label. Each pattern consists of the calls to each specific Macro, passing all needed data. Unlike the CTL output, a new “pattern type” macro will not be created for each pattern, and the pattern data will not be placed in a separate file and included with the “Include” statement.

To produce a flat or non-hierarchical set of vector data, set this keyword to 0 (default).

The following is an example of the structural STIL output:

```
STIL 1.0 ;

Header { ... }

// Parameter File Keyword Settings
// STIL_STRUCTURAL true ;

Signals {
   "IN"[3..0] In; "CLK" In; "SDI" In; "SE" In;
   "OUT"[3..0] Out; "SDO" Out;
}

SignalGroups {
   _pi_[6..0] = ’"IN"[3..0] + "CLK" + "SDI" + "SE"’;
   _po_[4..0] = ’"OUT"[3..0] + "SDO"’;
   PI_grp_0[5..0] = ’"IN"[3..0] + "SDI" + "SE"’;
   PI_grp_1 = ’"CLK"’;
   "_chain_SDI_" = ’"SDI"’ {ScanIn 10;}
   "_chain_SDO_" = ’"SDO"’ {ScanOut 10;}
}
```

```
Timing TRANSITION_timing {
   WaveformTable tset_tp_shift {
      Period '400ns' ;
      Waveforms {
         PI_grp_0 { 01X { '0ns' D/U/N; }}
         PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D;}}
         _po_ { LHXZ { '0ns' X; '50ns' l/h/X/t; '52ns' X;}}
      }
   }
   WaveformTable tset_tp_capture {
      Period '400ns' ;
      Waveforms {
         PI_grp_0 { 01X { '0ns' D/U/N; }}
         PI_grp_1 { 01 { '0ns' D; '100ns' D/U; '400ns' D;}}
         _po_ { LHXZ { '0ns' X; '50ns' l/h/X/t; '52ns' X;}}
      }
   }
}

ScanStructures { ScanChain chain { ... } }

Procedures {
   test_setup {
      W tset_tp_shift;
      V { "SE" = 0; }
   }
}

MacroDefs {
   load_unload_grp1 {
      W tset_tp_shift;
      V { "CLK" = 0; }
      Shift { V { "SE" = 1; "_chain_SDI_" = #;
                                    "_chain_SDO_" = #; "CLK" = 1; }
   }
}

capture {
   W tset_tp_shift;
   V { _pi_ = %; _po_ = %; PI_grp_1 = %; }
   }
}

PatternBurst scanpats { PatList { scan_test } }

PatternExec scanexec {
   Timing TRANSITION_timing;
   PatternBurst scanpats;
}
```

```
Pattern "scan_test" {
   Call test_setup ;
   Ann {* Begin chain test *}
   Ann {* Chain Pattern:0 Cycle:1 Loop:1 *}
   ...
   Ann {* End chain test *}
   Ann {* Pattern:0 Cycle:7 Loop:10 *}
   "pattern 0": Macro "load_unload_grp1" {
      "_chain_SDI_" = 1010001110;
      "_chain_SDO_" = XXXXXXXXXX;
}
Macro "capture" {
   _pi_ = X101011;
   _po_ = LLHHH;
PI_grp_1 = 1;
}

...
Ann {* Pattern:7 Cycle:24 Loop:52 *}
"pattern 0": Macro "load_unload_grp1" {
   "_chain_SDI_" = 1010010000;
   "_chain_SDO_" = HHLHHLLLLL;
}

Macro "capture" {
   _pi_ = X000011;
   _po_ = LHHLL;
   PI_grp_1 = 1;
}

last_unload: Macro "load_unload_grp1" {
   "_chain_SDI_" = 1010010000;
   "_chain_SDO_" = LHLLLHLLLH;
   }
}
```

STIL_TI_TITLE

Usage

```
STIL_TI_TITLE {1 | 0};
```

Description

Turns on a TI-specific title block in the STIL Header block. This title block contains all of the specific statements from the TITDL output file, and you can use the TITDL-specific keywords to change the fields in the title block. The default is to turn off the title block.

The TITDL-specific keywords include: TITDL_LIBRARY, TITDL_CUSTOMER, TITDL_PARTNUM, TITDL_SETNAME, TITDL_SETTYPE, TITDL_CHAIN_SETTYPE, and TITDL_REVISION.

STIL_TRIM_PULSE

Usage

```
STIL_TRIM_PULSE {1 | 0};
```

Description

Causes the input timing to drop the leading 0ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. To deactivate, set this keyword to 0 (default).

Examples follow:

```
Parameter file : default
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
            01 { '0ns' D; '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
IOclk { LHX { '0ns' X; '300ns' H/L/X; '400ns' X; }
            01 { '200ns' U/D; '300ns' D } }

Parameter file : STIL_TRIM_PULSE 1;
                                    STIL_NOMEASURE_CLOCK 1;
IOclk { 01 { '200ns' U/D; '300ns' D } }
```

STIL_VECTOR_ANN

Usage

```
STIL_VECTOR_ANN string %special_keyword string2;
```

Description

Specifies the vector types to be added to the output pattern file.

```
annotation "string %keywords string"
```

Note

You must not use the "%" symbol as the leading character to any string fragment in conjunction with the %special_keyword.

The STIL_SCAN_ANN1, STIL_PAT_ANN, and STIL_VECTOR_ANN parameter file keywords use a (%special_keyword) that is translated at the specific time and place the annotation statement is written. Table A-3 lists the available special keywords. Table A-4 lists the pattern types. Table A-5 lists the vector types.

Table A-3. STIL Special Keywords

Keyword	Description
%chain	The chain name of the current chain.
%cycle_num	The current ATPG cycle number.

Table A-3. STIL Special Keywords (cont.)

Keyword	Description
%length	The length of the current chain.
%pat_num	The current ATPG pattern number.
%pat_num_plus1	The current ATPG pattern number plus 1.
%pattern_type	The pattern type, for outputting annotations in the pattern file. It causes one of the pattern types in Table A-4 to be written into the pattern file.
%scan_in	The name of the Primary Input pin for the current chain.
%scan_out	The name of the Primary Output pin for the current chain.
%vector_type	This keyword causes one of the vector types in Table A-5 to be written into the pattern file.

Table A-4. STIL Pattern Types

basic	macrotest	ram_passthru
clock_po	multi_load	ram_sequential
clock_sequential		

Table A-5. STIL Vector Types

CLOCK_PROCEDURE	PRIMARY_CLOCKPO	SHADOW_CONTROL
CLOCK_PROCEDURE_RESTORE	PRIMARY_CLOCKPO_IDDQ	SHADOW_OBSERVE
CORE_ACCESS	PRIMARY_IDDQ	SHIFT
CORE_INST_ISOLATE	PRIMARY_MEASURE	SINGLE_SHIFT
CORE_INST_TEST	PRIMARY_MEASURE_IDDQ	SKEW_LOAD
LOAD_UNLOAD	SCAN_IN	SEQ_TRANSPARENT
MASTER_OBSERVE	SCAN_IO	TEST_SETUP
PRIMARY	SCAN_OUT	

STIL_VERG_ESC

Usage

```
STIL_VERG_ESC {1 | 0};
```

Description

Setting this keyword to 1 changes all vectored signal names to scalars. For example, the pin name “bus”[0] would change to “bus[0]”.

TITDL_CHAIN_SETTYPE

Usage

```
TITDL_CHAIN_SETTYPE string;
```

Description

Specifies the string used to set the chain type. If specified, the string is used in the output of Chain_Test-only patterns. If not specified, the default string is “SCANCHK”. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_CUSTOMER

Usage

```
TITDL_CUSTOMER string;
```

Description

Specifies the string the tool uses in the

```
CUSTOMER=string
```

statement in the header of the TITDL output file. If not specified, the default is TEXAS INSTRUMENTS. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_GROUP_TIMING

Usage

```
TITDL_GROUP_TIMING {1 | 0};
```

Description

Setting this keyword to 1 in conjunction with the “write_patterns -Titdl” command groups like timing information into a single statement. The default is off.

TITDL_KEEP_SCANOUT

Usage

```
TITDL_KEEP_SCANOUT {1 | 0};
```

Description

Setting this keyword to 1 causes all SCANOUT statements to be issued, even if all of the scanout values for a specific pattern/chain are X. The default is 0, which causes the tool to not keep scanout statements whose scanout values are all X when writing TITDL format vectors.

TITDL_LIBRARY

Usage

```
TITDL_LIBRARY string;
```

Description

Specifies the string that the tool uses in the

```
LIBRARY_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is GS20. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_PARTNUM

Usage

```
TITDL_PARTNUM string;
```

Description

Specifies the string the tool uses in the

```
TI_PART_NUMBER=string
```

statement in the header of the TITDL output file. If not specified, the default is F999999. To add this to the specific title block, see the parameter file keyword STIL_TI_TITLE.

TITDL_PSEUDO_PREFIX

Usage

```
TITDL_PSEUDO_PREFIX string;
```

Description

Specifies the string that is prepended to the name of the clock pin to create the name of the pseudo clock pin. If not specified, the default is “pseudo_”.

For example, this keyword changes the default pseudo prefix:

```
TITDL_PSEUDO_PREFIX ps_ ;
```

And this is the result of what the MUX_PIN statement would look like in the TITDL output. The clock pin is called “clock”, and the added pseudo clock pin is called “ps_clock” because of the use of the above parameter file keyword.

```
...
CONNECT P,
VAR =(in1, in2, clock, ps_clock, out1, out2),
DEFPIN=(IN 4, OUT 2);
PERIOD = 100 NS;
CLOCK VAR=(clock),
HOLD0 = 20 NS, HOLD1 = 10 NS, PATTERN = 010;
CLOCK VAR = (ps_clock),
HOLD0 = 60 NS, HOLD1 = 10 NS, PATTERN = 010;
MUX_PIN clock, VAR=(clock, ps_clock);
DELAY VAR = (in1),
OFFSET = 0 NS;
...
```

TITDL_REVISION

Usage

```
TITDL_REVISION string;
```

Description

Specifies the string that the tool uses in the

```
REVISION=string
```

statement in the header of the TITDL output file. If not specified, the default is 1.00. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_SETNAME

Usage

```
TITDL_SETNAME string;
```

Description

Specifies the string that the tool uses in the

```
PATTERN_SET_NAME=string
```

statement in the header of the TITDL output file. The string can be a maximum of twenty characters. If not specified, the default is the filename with no extension. To add this to the specific title block, use the STIL_TI_TITLE keyword.

TITDL_SETTYPE

Usage

```
TITDL_SETTYPE string;
```

Description

Specifies the string that the tool uses in the

```
PATTERN_SET_TYPE=string
```

statement in the header of the TITDL output file. If not specified, the default is SCAN. To add this to the specific title block, use the STIL_TI_TITLE keyword.

WGL_ADD_LASTX

Usage

```
WGL_ADD_LASTX {1 | 0};
```

Description

Specifies for the tool to add an additional vector to the end of the pattern set (when set to 1). This vector is a non-scan vector with all clocks turned off, all output pins set to “X” (not measuring), and all input pins set to the same value of the previous non-scan vector. To deactivate, set this keyword to 0 (the default).

WGL_ALT_BIDI

Usage

```
WGL_ALT_BIDI {1 | 0};
```

Description

Changes the syntax of how bidirectional signals are represented in the vector pin list, and the actual states in the vector. The default representation (keyword set to 0) specifies the bidirectional signal name, a colon, then an I or O, for the direction of that pin. There must be two entries for the bidirectional pin, one specifying the input side and one specifying the output side. The vector lists a separate state for each.

Setting this keyword to 1 changes the representation so that the pin list lists only one entry for the bidirectional pin with no direction, but with two states. Every state in the actual vector is currently separated by spaces. With the addition of this WGL_ALT_BIDI 1 setting, the states for the bidirectional pin will be listed side-by-side with no space separation. The input state will be listed first.

WGL_ALT_BIDI set to 0 (default) — The format of the vector order pin list and vector is as follows:

```
pattern test("A","B","C","BiDi1:I","E","F","G","BiDi1:O")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 - 0 1 0 1];
```

WGL_ALT_BIDI set to 1 — The vector order pinlist and vector are changed to the following:

```
pattern test("A","B","C","BiDi1","E","F","G")
{Pattern 0 Cycle 0 Loop 0}
vector(+,TP1):=[1 0 1 -1 0 1 0];
```

WGL_ALT_VECT_ANN

Usage

```
WGL_ALT_VECT_ANN {1 | 0};
```

Description

Specifies alternate annotation statements for beginnings of patterns and vector boundaries when this keyword is set to 1. These alternate annotations can help the WGL file be processed by other third party tools. The default is 0.

WGL_EDGE_STROBE

Usage

```
WGL_EDGE_STROBE {1 | 0};
```

Description

The default value of this keyword is 0. This keyword is in effect only when the strobe_window is set to 0 using the "set strobe_window time" statement in the procedure file.

The WGL output specifies edge strobes by using the edge qualifier on the strobe times specified in timeplate definitions. Additionally, an output value showing the "X' state appears after the strobe time.

If the strobe_window is set to 0, then this additional output value showing the "X' state after the strobe time can be eliminated from the outputted WGL by setting this keyword to 1.

The following is an example of an edge strobe statement without and with using WGL_EDGE_STROBE.

WGL_EDGE_STROBE set to 0 (default):

```
"outpin":=output[0ns:X,300ns:Q'edge,350ns:X];
```

WGL_EDGE_STROBE set to 1

```
"outpin":=output[0ns:X,300ns:Q'edge];
```

WGL_FULL_CHAIN

Usage

```
WGL_FULL_CHAIN {1 | 0};
```

Description

When set to 1, this keyword alters the scancell and scanchain blocks of the WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts (number of apply shifts). The group statements only reflect what is loaded minus the post shift. The scanstate and group have to match. To deactivate, and for backwards compatibility, issue a 0 (false).

Examples follow.

Original WGL File (WGL_FULL_CHAIN 0)

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u12", "u11", !, "u10", "u23", "u22", "sdo"];
    end
 ...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (11010);
 ...
    end
 ...
```

WGL_FULL_CHAIN 1

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
"sdo"];
    end
 ...
```

```
        scanstate
            OREG1_sts000000_000002 := groupgrp1REG1 (XXXXX);
            IREG1_sts000000_000002 := groupgrp1REG1 (11010);
    ...
        end
    ...
```

WGL_FULL_SCANGROUP

Usage

```
WGL_FULL_SCANGROUP {1 | 0};
```

Description

Creates the Scan sections with the full chain (when set to 1). This keyword alters the scainchain, scancell, and scanstate blocks in WGL to reflect all cells in the chain, regardless of the number of pre- or post-shifts. Setting the keyword to 0, deactivates this feature.

The following is an example of WGL code when WGL_FULL_SCANGROUP and WGL_FULL_CHAIN are both set to 1:

```
...
    scancell
        { Group: grp1 }
        { Chain: REG1 }
        "u22"; "u23"; "u10"; "u11"; "u12"; "u13";
        groupgrp1REG1 ["u22", "u23", "u10", "u11", "u12", "u13"];
    end
    scanchain
        { Group: grp1 }
        REG1_chain[ "sdi", "u13", "u12", "u11", !, "u10", "u23", "u22",
                                                                "sdo"];
    end
...
    scanstate
        OREG1_sts000000_000002 := groupgrp1REG1 (XXXXXX);
        IREG1_sts000000_000002 := groupgrp1REG1 (X1010 0);
...
    end
...
```

WGL_GROUP_PIN

Usage

```
WGL_GROUP_PIN {1 | 0};
```

Description

Specifies for the WGL output to create signal groups _PI_, _PO_, and _BIDI_ (if there are bidi's) when set to 1. The WGL code then uses these groups in the pattern statement and in actual patterns for grouping states. To deactivate, set this keyword to 0.

WGL_INV_SC

Usage

```
WGL_INV_SC {1 | 0};
```

Description

Removes inversion data from scan chain definitions and specifies scan data as it would appear to be shifted in, instead of parallel deposited. According to the WGL data model, a WGL file that uses this option no longer correctly matches the design netlist. However, some third-party tools work faster with the inversion data removed. When set to 0 (default), this keyword leaves the inversion data intact.

WGL_NO_SCANX

Usage

```
WGL_NO_SCANX {1 | 0};
```

Description

Replaces all X scan load values in the WGL pattern output with 0 scan load values. When set to 0 (default), this keyword deactivates this feature.

WGL_NOMEASURE_CLOCK

Usage

```
WGL_NOMEASURE_CLOCK {1 | 0};
```

Description

Drops the measure timing from the timing statement for bidirectional clocks and also verifies that this pin is never measured. If the pin is measured within the patterns, the output generates an error and returns to the command prompt. Setting this keyword to 0 (default), deactivates this feature.

Note that this keyword can cause some WGL readers to have problems reading the file. The tool issues a warning if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow:

```
Parameter file : default
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];

Parameter file : WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];

Parameter file : WGL_TRIM_PULSE 1;
                                        WGL_NOMEASURE_CLOCK 1;
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_ONE_ILLINOIS

Usage

```
WGL_ONE_ILLINOIS {1 | 0};
```

Description

Causes the WGL output to have only one scan state defined per pattern for multiple scan chains that share a single scan input pin. In other words, this keyword forces the tool to produce a single scan state for chains that share a common scan input pin (set to 1).

The default behavior when this keyword is set to 0 is to have scan states defined for each scan chain, whether or not they share scan input pins. The default for scan states when multiple scan chains share a common scan input is to list the scan states for each cell in each chain, the same as if the chains did not share a scan input.

WGL_PATTERN_NAME

Usage

WGL_PATTERN_NAME *string*;

Description

Changes the default name of the WGL pattern block from “Chain_scan_test” to the value of the string.

WGL_TRIM_LEADING_P

Usage

```
WGL_TRIM_LEADING_P {1 | 0};
```

Description

Removes the leading 0ns:P timing event when a channel uses timing variables (keyword set to 1). For example, the default behavior produces a timeplate channel that looks like this:

```
"pi_sel_smcard" := input[0ns:P, fd_force:S];
```

When WGL_TRIM_LEADING_P is set to 1 in the parameter file, the timeplate channel looks like this:

```
"pi_sel_smcard" := input[fd_force:S];
```

WGL_TRIM_PULSE

Usage

```
WGL_TRIM_PULSE {1 | 0};
```

Description

Causes the input timing to drop the leading 0 ns force to the off state of the clock (keyword set to 1). Removing the initial force to the off state of a clock may not be appropriate for all testers and translation processes, and you need to understand all downstream processes that use the resultant files. Setting this keyword to 0 deactivates this feature.

Note that this keyword generates illegal syntax in the WGL file, which may cause some WGL readers to have problems reading the file. The tool issues a warning message if the keyword is specified in the parameter file and if there is at least one signal using the altered syntax.

Examples follow.

Parameter file : default

```
IOclk := input [ 0ns:D, 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```

Parameter file : WGL_TRIM_PULSE 1;

```
IOclk := input [ 200ns:S, 300ns:D ];
IOclk := output [ 0ns:X, 300ns:Q, 400ns:X ];
```

Parameter file : WGL_TRIM_PULSE 1;
qWGL_NOMEASURE_CLOCK 1;

```
IOclk := input [ 200ns:S, 300ns:D ];
```

WGL_VECTOR_ANN

Usage

```
WGL_VECTOR_ANN {1 | 0};
```

Description

Adds an annotation statement to be generated in the pattern file to denote the vector type. Setting this keyword to 0 deactivates this feature.

WGL_VERG_ESC

Usage

```
WGL_VERG_ESC {1 | 0};
```

Description

Specifies for buses to be expanded in the declaration sections of the WGL output. Setting this keyword to 0 deactivates this feature.

WGL_VTRAN_PADSC

Usage

```
WGL_VTRAN_PADSC {1 | 0};
```

Description

Pads all scan data to be same length (keyword set to 1). Padding all scan data to be the same length violates the WGL syntax according to the TDS WGL documentation. However, some third-party tools work better with the scan data padded. Setting this keyword to 0 deactivates this feature.

Appendix B
Getting Help

There are several ways to get help when setting up and using Tessent software tools. Depending on your need, help is available from documentation, online command help, and Mentor Graphics Support.

Documentation

The Tessent software tree includes a complete set of documentation and help files in PDF format. Although you can view this documentation with any PDF reader, if you are viewing documentation on a Linux file server, you must use only Adobe® Reader® versions 8 or 9, and you must set one of these versions as the default using the MGC_PDF_READER variable in your *mgc_doc_options.ini* file.

For more information, refer to "Specifying Documentation System Defaults" in the *Managing Mentor Graphics Tessent Software* manual.

You can download a free copy of the latest Adobe Reader from this location:

http://get.adobe.com/reader

You can access the documentation in the following ways:

- **Shell Command** — On Linux platforms, enter **mgcdocs** at the shell prompt or invoke a Tessent tool with the -Manual invocation switch. This option is available only with Tessent Shell and the following classic point tools: Tessent FastScan, Tessent TestKompress, Tessent Diagnosis, and DFTAdvisor.
- **File System** — Access the Tessent bookcase directly from your file system, without invoking a Tessent tool. From your product installation, invoke Adobe Reader on the following file:

  ```
  $MGC_DFT/doc/pdfdocs/_bk_tessent.pdf
  ```

- **Application Online Help** — You can get contextual online help within most Tessent tools by using the "help -manual" tool command:

  > **help dofile -manual**

  This command opens the appropriate reference manual at the "dofile" command description.

Mentor Graphics Support

Mentor Graphics software support includes software enhancements, access to comprehensive online services with SupportNet, and the optional On-Site Mentoring service.

For details, refer to this page:

http://supportnet.mentor.com/about

If you have questions about a software release, you can log in to SupportNet and search thousands of technical solutions, view documentation, or open a Service Request online:

http://supportnet.mentor.com

If your site is under current support and you do not have a SupportNet login, you can register for SupportNet by filling out a short form here:

http://supportnet.mentor.com/user/register.cfm

All customer support contact information is available here:

http://supportnet.mentor.com/contacts/supportcenters/index.cfm

Third-Party Information

For information about third-party software included with this release of Tessent products, refer to the *Third-Party Software for Tessent Products*.

End-User License Agreement

The latest version of the End-User License Agreement is available on-line at:
www.mentor.com/eula

IMPORTANT INFORMATION

USE OF ALL SOFTWARE IS SUBJECT TO LICENSE RESTRICTIONS. CAREFULLY READ THIS LICENSE AGREEMENT BEFORE USING THE PRODUCTS. USE OF SOFTWARE INDICATES CUSTOMER'S COMPLETE AND UNCONDITIONAL ACCEPTANCE OF THE TERMS AND CONDITIONS SET FORTH IN THIS AGREEMENT. ANY ADDITIONAL OR DIFFERENT PURCHASE ORDER TERMS AND CONDITIONS SHALL NOT APPLY.

END-USER LICENSE AGREEMENT ("Agreement")

This is a legal agreement concerning the use of Software (as defined in Section 2) and hardware (collectively "Products") between the company acquiring the Products ("Customer"), and the Mentor Graphics entity that issued the corresponding quotation or, if no quotation was issued, the applicable local Mentor Graphics entity ("Mentor Graphics"). Except for license agreements related to the subject matter of this license agreement which are physically signed by Customer and an authorized representative of Mentor Graphics, this Agreement and the applicable quotation contain the parties' entire understanding relating to the subject matter and supersede all prior or contemporaneous agreements. If Customer does not agree to these terms and conditions, promptly return or, in the case of Software received electronically, certify destruction of Software and all accompanying items within five days after receipt of Software and receive a full refund of any license fee paid.

1. **ORDERS, FEES AND PAYMENT.**

   1.1. To the extent Customer (or if agreed by Mentor Graphics, Customer's appointed third party buying agent) places and Mentor Graphics accepts purchase orders pursuant to this Agreement (each an "Order"), each Order will constitute a contract between Customer and Mentor Graphics, which shall be governed solely and exclusively by the terms and conditions of this Agreement, any applicable addenda and the applicable quotation, whether or not those documents are referenced on the Order. Any additional or conflicting terms and conditions appearing on an Order or presented in any electronic portal or automated order management system, whether or not required to be electronically accepted, will not be effective unless agreed in writing and physically signed by an authorized representative of Customer and Mentor Graphics.

   1.2. Amounts invoiced will be paid, in the currency specified on the applicable invoice, within 30 days from the date of such invoice. Any past due invoices will be subject to the imposition of interest charges in the amount of one and one-half percent per month or the applicable legal rate currently in effect, whichever is lower. Prices do not include freight, insurance, customs duties, taxes or other similar charges, which Mentor Graphics will state separately in the applicable invoice. Unless timely provided with a valid certificate of exemption or other evidence that items are not taxable, Mentor Graphics will invoice Customer for all applicable taxes including, but not limited to, VAT, GST, sales tax, consumption tax and service tax. Customer will make all payments free and clear of, and without reduction for, any withholding or other taxes; any such taxes imposed on payments by Customer hereunder will be Customer's sole responsibility. If Customer appoints a third party to place purchase orders and/or make payments on Customer's behalf, Customer shall be liable for payment under Orders placed by such third party in the event of default.

   1.3. All Products are delivered FCA factory (Incoterms 2010), freight prepaid and invoiced to Customer, except Software delivered electronically, which shall be deemed delivered when made available to Customer for download. Mentor Graphics retains a security interest in all Products delivered under this Agreement, to secure payment of the purchase price of such Products, and Customer agrees to sign any documents that Mentor Graphics determines to be necessary or convenient for use in filing or perfecting such security interest. Mentor Graphics' delivery of Software by electronic means is subject to Customer's provision of both a primary and an alternate e-mail address.

2. **GRANT OF LICENSE.** The software installed, downloaded, or otherwise acquired by Customer under this Agreement, including any updates, modifications, revisions, copies, documentation and design data ("Software") are copyrighted, trade secret and confidential information of Mentor Graphics or its licensors, who maintain exclusive title to all Software and retain all rights not expressly granted by this Agreement. Mentor Graphics grants to Customer, subject to payment of applicable license fees, a nontransferable, nonexclusive license to use Software solely: (a) in machine-readable, object-code form (except as provided in Subsection 5.2); (b) for Customer's internal business purposes; (c) for the term of the license; and (d) on the computer hardware and at the site authorized by Mentor Graphics. A site is restricted to a one-half mile (800 meter) radius. Customer may have Software temporarily used by an employee for telecommuting purposes from locations other than a Customer office, such as the employee's residence, an airport or hotel, provided that such employee's primary place of employment is the site where the Software is authorized for use. Mentor Graphics' standard policies and programs, which vary depending on Software, license fees paid or services purchased, apply to the following: (a) relocation of Software; (b) use of Software, which may be limited, for example, to execution of a single session by a single user on the authorized hardware or for a restricted period of time (such limitations may be technically implemented through the use of authorization codes or similar devices); and (c) support services provided, including eligibility to receive telephone support, updates, modifications, and revisions. For the avoidance of doubt, if Customer provides any feedback or requests any change or enhancement to Products, whether in the course of receiving support or consulting services, evaluating Products, performing beta testing or otherwise, any inventions, product improvements, modifications or developments made by Mentor Graphics (at Mentor Graphics' sole discretion) will be the exclusive property of Mentor Graphics.

3. **ESC SOFTWARE.** If Customer purchases a license to use development or prototyping tools of Mentor Graphics' Embedded Software Channel ("ESC"), Mentor Graphics grants to Customer a nontransferable, nonexclusive license to reproduce and distribute executable files created using ESC compilers, including the ESC run-time libraries distributed with ESC C and C++ compiler Software that are

linked into a composite program as an integral part of Customer's compiled computer program, provided that Customer distributes these files only in conjunction with Customer's compiled computer program. Mentor Graphics does NOT grant Customer any right to duplicate, incorporate or embed copies of Mentor Graphics' real-time operating systems or other embedded software products into Customer's products or applications without first signing or otherwise agreeing to a separate agreement with Mentor Graphics for such purpose.

4. **BETA CODE.**

4.1. Portions or all of certain Software may contain code for experimental testing and evaluation (which may be either alpha or beta, collectively "Beta Code"), which may not be used without Mentor Graphics' explicit authorization. Upon Mentor Graphics' authorization, Mentor Graphics grants to Customer a temporary, nontransferable, nonexclusive license for experimental use to test and evaluate the Beta Code without charge for a limited period of time specified by Mentor Graphics. Mentor Graphics may choose, at its sole discretion, not to release Beta Code commercially in any form.

4.2. If Mentor Graphics authorizes Customer to use the Beta Code, Customer agrees to evaluate and test the Beta Code under normal conditions as directed by Mentor Graphics. Customer will contact Mentor Graphics periodically during Customer's use of the Beta Code to discuss any malfunctions or suggested improvements. Upon completion of Customer's evaluation and testing, Customer will send to Mentor Graphics a written evaluation of the Beta Code, including its strengths, weaknesses and recommended improvements.

4.3. Customer agrees to maintain Beta Code in confidence and shall restrict access to the Beta Code, including the methods and concepts utilized therein, solely to those employees and Customer location(s) authorized by Mentor Graphics to perform beta testing. Customer agrees that any written evaluations and all inventions, product improvements, modifications or developments that Mentor Graphics conceived or made during or subsequent to this Agreement, including those based partly or wholly on Customer's feedback, will be the exclusive property of Mentor Graphics. Mentor Graphics will have exclusive rights, title and interest in all such property. The provisions of this Subsection 4.3 shall survive termination of this Agreement.

5. **RESTRICTIONS ON USE.**

5.1. Customer may copy Software only as reasonably necessary to support the authorized use. Each copy must include all notices and legends embedded in Software and affixed to its medium and container as received from Mentor Graphics. All copies shall remain the property of Mentor Graphics or its licensors. Customer shall maintain a record of the number and primary location of all copies of Software, including copies merged with other software, and shall make those records available to Mentor Graphics upon request. Customer shall not make Products available in any form to any person other than Customer's employees and on-site contractors, excluding Mentor Graphics competitors, whose job performance requires access and who are under obligations of confidentiality. Customer shall take appropriate action to protect the confidentiality of Products and ensure that any person permitted access does not disclose or use Products except as permitted by this Agreement. Customer shall give Mentor Graphics written notice of any unauthorized disclosure or use of the Products as soon as Customer becomes aware of such unauthorized disclosure or use. Except as otherwise permitted for purposes of interoperability as specified by applicable and mandatory local law, Customer shall not reverse-assemble, reverse-compile, reverse-engineer or in any way derive any source code from Software. Log files, data files, rule files and script files generated by or for the Software (collectively "Files"), including without limitation files containing Standard Verification Rule Format ("SVRF") and Tcl Verification Format ("TVF") which are Mentor Graphics' trade secret and proprietary syntaxes for expressing process rules, constitute or include confidential information of Mentor Graphics. Customer may share Files with third parties, excluding Mentor Graphics competitors, provided that the confidentiality of such Files is protected by written agreement at least as well as Customer protects other information of a similar nature or importance, but in any case with at least reasonable care. Customer may use Files containing SVRF or TVF only with Mentor Graphics products. Under no circumstances shall Customer use Products or Files or allow their use for the purpose of developing, enhancing or marketing any product that is in any way competitive with Products, or disclose to any third party the results of, or information pertaining to, any benchmark.

5.2. If any Software or portions thereof are provided in source code form, Customer will use the source code only to correct software errors and enhance or modify the Software for the authorized use. Customer shall not disclose or permit disclosure of source code, in whole or in part, including any of its methods or concepts, to anyone except Customer's employees or on-site contractors, excluding Mentor Graphics competitors, with a need to know. Customer shall not copy or compile source code in any manner except to support this authorized use.

5.3. Customer may not assign this Agreement or the rights and duties under it, or relocate, sublicense, or otherwise transfer the Products, whether by operation of law or otherwise ("Attempted Transfer"), without Mentor Graphics' prior written consent and payment of Mentor Graphics' then-current applicable relocation and/or transfer fees. Any Attempted Transfer without Mentor Graphics' prior written consent shall be a material breach of this Agreement and may, at Mentor Graphics' option, result in the immediate termination of the Agreement and/or the licenses granted under this Agreement. The terms of this Agreement, including without limitation the licensing and assignment provisions, shall be binding upon Customer's permitted successors in interest and assigns.

5.4. The provisions of this Section 5 shall survive the termination of this Agreement.

6. **SUPPORT SERVICES.** To the extent Customer purchases support services, Mentor Graphics will provide Customer with updates and technical support for the Products, at the Customer site(s) for which support is purchased, in accordance with Mentor Graphics' then current End-User Support Terms located at http://supportnet.mentor.com/supportterms.

7. **LIMITED WARRANTY.**

7.1. Mentor Graphics warrants that during the warranty period its standard, generally supported Products, when properly installed, will substantially conform to the functional specifications set forth in the applicable user manual. Mentor Graphics does not warrant that Products will meet Customer's requirements or that operation of Products will be uninterrupted or error free. The warranty period is 90 days starting on the 15th day after delivery or upon installation, whichever first occurs. Customer must notify Mentor

Graphics in writing of any nonconformity within the warranty period. For the avoidance of doubt, this warranty applies only to the initial shipment of Software under an Order and does not renew or reset, for example, with the delivery of (a) Software updates or (b) authorization codes or alternate Software under a transaction involving Software re-mix. This warranty shall not be valid if Products have been subject to misuse, unauthorized modification, improper installation or Customer is not in compliance with this Agreement. MENTOR GRAPHICS' ENTIRE LIABILITY AND CUSTOMER'S EXCLUSIVE REMEDY SHALL BE, AT MENTOR GRAPHICS' OPTION, EITHER (A) REFUND OF THE PRICE PAID UPON RETURN OF THE PRODUCTS TO MENTOR GRAPHICS OR (B) MODIFICATION OR REPLACEMENT OF THE PRODUCTS THAT DO NOT MEET THIS LIMITED WARRANTY. MENTOR GRAPHICS MAKES NO WARRANTIES WITH RESPECT TO: (A) SERVICES; (B) PRODUCTS PROVIDED AT NO CHARGE; OR (C) BETA CODE; ALL OF WHICH ARE PROVIDED "AS IS."

7.2. THE WARRANTIES SET FORTH IN THIS SECTION 7 ARE EXCLUSIVE. NEITHER MENTOR GRAPHICS NOR ITS LICENSORS MAKE ANY OTHER WARRANTIES EXPRESS, IMPLIED OR STATUTORY, WITH RESPECT TO PRODUCTS PROVIDED UNDER THIS AGREEMENT. MENTOR GRAPHICS AND ITS LICENSORS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF INTELLECTUAL PROPERTY.

8. **LIMITATION OF LIABILITY.** EXCEPT WHERE THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES (INCLUDING LOST PROFITS OR SAVINGS) WHETHER BASED ON CONTRACT, TORT OR ANY OTHER LEGAL THEORY, EVEN IF MENTOR GRAPHICS OR ITS LICENSORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. IN NO EVENT SHALL MENTOR GRAPHICS' OR ITS LICENSORS' LIABILITY UNDER THIS AGREEMENT EXCEED THE AMOUNT RECEIVED FROM CUSTOMER FOR THE HARDWARE, SOFTWARE LICENSE OR SERVICE GIVING RISE TO THE CLAIM. IN THE CASE WHERE NO AMOUNT WAS PAID, MENTOR GRAPHICS AND ITS LICENSORS SHALL HAVE NO LIABILITY FOR ANY DAMAGES WHATSOEVER. THE PROVISIONS OF THIS SECTION 8 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

9. **HAZARDOUS APPLICATIONS.** CUSTOMER ACKNOWLEDGES IT IS SOLELY RESPONSIBLE FOR TESTING ITS PRODUCTS USED IN APPLICATIONS WHERE THE FAILURE OR INACCURACY OF ITS PRODUCTS MIGHT RESULT IN DEATH OR PERSONAL INJURY ("HAZARDOUS APPLICATIONS"). EXCEPT TO THE EXTENT THIS EXCLUSION OR RESTRICTION OF LIABILITY WOULD BE VOID OR INEFFECTIVE UNDER APPLICABLE LAW, IN NO EVENT SHALL MENTOR GRAPHICS OR ITS LICENSORS BE LIABLE FOR ANY DAMAGES RESULTING FROM OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 9 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

10. **INDEMNIFICATION.** CUSTOMER AGREES TO INDEMNIFY AND HOLD HARMLESS MENTOR GRAPHICS AND ITS LICENSORS FROM ANY CLAIMS, LOSS, COST, DAMAGE, EXPENSE OR LIABILITY, INCLUDING ATTORNEYS' FEES, ARISING OUT OF OR IN CONNECTION WITH THE USE OF MENTOR GRAPHICS PRODUCTS IN OR FOR HAZARDOUS APPLICATIONS. THE PROVISIONS OF THIS SECTION 10 SHALL SURVIVE THE TERMINATION OF THIS AGREEMENT.

11. **INFRINGEMENT.**

11.1. Mentor Graphics will defend or settle, at its option and expense, any action brought against Customer in the United States, Canada, Japan, or member state of the European Union which alleges that any standard, generally supported Product acquired by Customer hereunder infringes a patent or copyright or misappropriates a trade secret in such jurisdiction. Mentor Graphics will pay costs and damages finally awarded against Customer that are attributable to such action. Customer understands and agrees that as conditions to Mentor Graphics' obligations under this section Customer must: (a) notify Mentor Graphics promptly in writing of the action; (b) provide Mentor Graphics all reasonable information and assistance to settle or defend the action; and (c) grant Mentor Graphics sole authority and control of the defense or settlement of the action.

11.2. If a claim is made under Subsection 11.1 Mentor Graphics may, at its option and expense: (a) replace or modify the Product so that it becomes noninfringing; (b) procure for Customer the right to continue using the Product; or (c) require the return of the Product and refund to Customer any purchase price or license fee paid, less a reasonable allowance for use.

11.3. Mentor Graphics has no liability to Customer if the action is based upon: (a) the combination of Software or hardware with any product not furnished by Mentor Graphics; (b) the modification of the Product other than by Mentor Graphics; (c) the use of other than a current unaltered release of Software; (d) the use of the Product as part of an infringing process; (e) a product that Customer makes, uses, or sells; (f) any Beta Code or Product provided at no charge; (g) any software provided by Mentor Graphics' licensors who do not provide such indemnification to Mentor Graphics' customers; or (h) infringement by Customer that is deemed willful. In the case of (h), Customer shall reimburse Mentor Graphics for its reasonable attorney fees and other costs related to the action.

11.4. THIS SECTION 11 IS SUBJECT TO SECTION 8 ABOVE AND STATES THE ENTIRE LIABILITY OF MENTOR GRAPHICS AND ITS LICENSORS, AND CUSTOMER'S SOLE AND EXCLUSIVE REMEDY, FOR DEFENSE, SETTLEMENT AND DAMAGES, WITH RESPECT TO ANY ALLEGED PATENT OR COPYRIGHT INFRINGEMENT OR TRADE SECRET MISAPPROPRIATION BY ANY PRODUCT PROVIDED UNDER THIS AGREEMENT.

12. **TERMINATION AND EFFECT OF TERMINATION.**

12.1. If a Software license was provided for limited term use, such license will automatically terminate at the end of the authorized term. Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement immediately upon written notice if Customer: (a) exceeds the scope of the license or otherwise fails to comply with the licensing or confidentiality provisions of this Agreement, or (b) becomes insolvent, files a bankruptcy petition, institutes proceedings for liquidation or winding up or enters into an agreement to assign its assets for the benefit of creditors. For any other material breach of any provision of this Agreement, Mentor Graphics may terminate this Agreement and/or any license granted under this Agreement upon 30 days written notice if Customer fails to cure the breach within the 30 day notice period. Termination of this Agreement or

any license granted hereunder will not affect Customer's obligation to pay for Products shipped or licenses granted prior to the termination, which amounts shall be payable immediately upon the date of termination.

12.2. Upon termination of this Agreement, the rights and obligations of the parties shall cease except as expressly set forth in this Agreement. Upon termination, Customer shall ensure that all use of the affected Products ceases, and shall return hardware and either return to Mentor Graphics or destroy Software in Customer's possession, including all copies and documentation, and certify in writing to Mentor Graphics within ten business days of the termination date that Customer no longer possesses any of the affected Products or copies of Software in any form.

13. **EXPORT.** The Products provided hereunder are subject to regulation by local laws and United States ("U.S.") government agencies, which prohibit export, re-export or diversion of certain products, information about the products, and direct or indirect products thereof, to certain countries and certain persons. Customer agrees that it will not export or re-export Products in any manner without first obtaining all necessary approval from appropriate local and U.S. government agencies. If Customer wishes to disclose any information to Mentor Graphics that is subject to any U.S. or other applicable export restrictions, including without limitation the U.S. International Traffic in Arms Regulations (ITAR) or special controls under the Export Administration Regulations (EAR), Customer will notify Mentor Graphics personnel, in advance of each instance of disclosure, that such information is subject to such export restrictions.

14. **U.S. GOVERNMENT LICENSE RIGHTS.** Software was developed entirely at private expense. The parties agree that all Software is commercial computer software within the meaning of the applicable acquisition regulations. Accordingly, pursuant to U.S. FAR 48 CFR 12.212 and DFAR 48 CFR 227.7202, use, duplication and disclosure of the Software by or for the U.S. government or a U.S. government subcontractor is subject solely to the terms and conditions set forth in this Agreement, which shall supersede any conflicting terms or conditions in any government order document, except for provisions which are contrary to applicable mandatory federal laws.

15. **THIRD PARTY BENEFICIARY.** Mentor Graphics Corporation, Mentor Graphics (Ireland) Limited, Microsoft Corporation and other licensors may be third party beneficiaries of this Agreement with the right to enforce the obligations set forth herein.

16. **REVIEW OF LICENSE USAGE.** Customer will monitor the access to and use of Software. With prior written notice and during Customer's normal business hours, Mentor Graphics may engage an internationally recognized accounting firm to review Customer's software monitoring system and records deemed relevant by the internationally recognized accounting firm to confirm Customer's compliance with the terms of this Agreement or U.S. or other local export laws. Such review may include FlexNet (or successor product) report log files that Customer shall capture and provide at Mentor Graphics' request. Customer shall make records available in electronic format and shall fully cooperate with data gathering to support the license review. Mentor Graphics shall bear the expense of any such review unless a material non-compliance is revealed. Mentor Graphics shall treat as confidential information all information gained as a result of any request or review and shall only use or disclose such information as required by law or to enforce its rights under this Agreement. The provisions of this Section 16 shall survive the termination of this Agreement.

17. **CONTROLLING LAW, JURISDICTION AND DISPUTE RESOLUTION.** The owners of certain Mentor Graphics intellectual property licensed under this Agreement are located in Ireland and the U.S. To promote consistency around the world, disputes shall be resolved as follows: excluding conflict of laws rules, this Agreement shall be governed by and construed under the laws of the State of Oregon, U.S., if Customer is located in North or South America, and the laws of Ireland if Customer is located outside of North or South America. All disputes arising out of or in relation to this Agreement shall be submitted to the exclusive jurisdiction of the courts of Portland, Oregon when the laws of Oregon apply, or Dublin, Ireland when the laws of Ireland apply. Notwithstanding the foregoing, all disputes in Asia arising out of or in relation to this Agreement shall be resolved by arbitration in Singapore before a single arbitrator to be appointed by the chairman of the Singapore International Arbitration Centre ("SIAC") to be conducted in the English language, in accordance with the Arbitration Rules of the SIAC in effect at the time of the dispute, which rules are deemed to be incorporated by reference in this section. Nothing in this section shall restrict Mentor Graphics' right to bring an action (including for example a motion for injunctive relief) against Customer in the jurisdiction where Customer's place of business is located. The United Nations Convention on Contracts for the International Sale of Goods does not apply to this Agreement.

18. **SEVERABILITY.** If any provision of this Agreement is held by a court of competent jurisdiction to be void, invalid, unenforceable or illegal, such provision shall be severed from this Agreement and the remaining provisions will remain in full force and effect.

19. **MISCELLANEOUS.** This Agreement contains the parties' entire understanding relating to its subject matter and supersedes all prior or contemporaneous agreements. Some Software may contain code distributed under a third party license agreement that may provide additional rights to Customer. Please see the applicable Software documentation for details. This Agreement may only be modified in writing, signed by an authorized representative of each party. Waiver of terms or excuse of breach must be in writing and shall not constitute subsequent consent, waiver or excuse.

Rev. 140201, Part No. 258976