

Command Reference for Encounter® RTL Compiler

Product Version 11.2
June 2012

© 2003-2012 Cadence Design Systems, Inc. All rights reserved.

Printed in the United States of America.

Cadence Design Systems, Inc. (Cadence), 2655 Seely Ave., San Jose, CA 95134, USA.

Open SystemC, Open SystemC Initiative, OSCI, SystemC, and SystemC Initiative are trademarks or registered trademarks of Open SystemC Initiative, Inc. in the United States and other countries and are used with permission.

Trademarks: Trademarks and service marks of Cadence Design Systems, Inc. contained in this document are attributed to Cadence with the appropriate symbol. For queries regarding Cadence's trademarks, contact the corporate legal department at the address shown above or call 800.862.4522. All other trademarks are the property of their respective holders.

Restricted Permission: This publication is protected by copyright law and international treaties and contains trade secrets and proprietary information owned by Cadence. Unauthorized reproduction or distribution of this publication, or any portion of it, may result in civil and criminal penalties. Except as specified in this permission statement, this publication may not be copied, reproduced, modified, published, uploaded, posted, transmitted, or distributed in any way, without prior written permission from Cadence. Unless otherwise agreed to by Cadence in writing, this statement grants Cadence customers permission to print one (1) hard copy of this publication subject to the following conditions:

1. The publication may be used only in accordance with a written agreement between Cadence and its customer.
2. The publication may not be modified in any way.
3. Any authorized copy of the publication or portion thereof must include all original copyright, trademark, and other proprietary notices and this permission statement.
4. The information contained in this document cannot be used in the development of like products or software, whether for internal or external use, and shall not be used for the benefit of any other party, whether or not for consideration.

Patents: Cadence Product Encounter™ RTL Compiler described in this document, is protected by U.S. Patents [5,892,687]; [6,470,486]; 6,772,398]; [6,772,399]; [6,807,651]; [6,832,357]; [7,007,247] and [8,127,260]

Disclaimer: Information in this publication is subject to change without notice and does not represent a commitment on the part of Cadence. Except as may be explicitly set forth in such agreement, Cadence does not make, and expressly disclaims, any representations or warranties as to the completeness, accuracy or usefulness of the information contained in this document. Cadence does not warrant that use of such information will not infringe any third party rights, nor does Cadence assume any liability for damages or costs of any kind that may result from use of such information.

Restricted Rights: Use, duplication, or disclosure by the Government is subject to restrictions as set forth in FAR52.227-14 and DFAR252.227-7013 et seq. or its successor

Contents

<u>Alphabetical List of Commands</u>	19
<u>Preface</u>	25
<u>About This Manual</u>	26
<u>Additional References</u>	26
<u>How to Use the Documentation Set</u>	27
<u>Reporting Problems or Errors in Manuals</u>	28
<u>Customer Support</u>	29
<u>Cadence Online Support</u>	29
<u>Other Support Offerings</u>	29
<u>Messages</u>	30
<u>Man Pages</u>	31
<u>Command-Line Help</u>	32
<u>Getting the Syntax for a Command</u>	32
<u>Getting the Syntax for an Attribute</u>	32
<u>Searching for Attributes</u>	33
<u>Searching For Commands When You Are Unsure of the Name</u>	33
<u>Documentation Conventions</u>	34
<u>Text Command Syntax</u>	34
 <u>1</u>	
<u>Navigation</u>	35
<u>basename</u>	36
<u>cd</u>	37
<u>dirname</u>	39
<u>dirs</u>	40
<u>filter</u>	41
<u>find</u>	44
<u>inout_mate</u>	48
<u>ll</u>	49

Command Reference for Encounter RTL Compiler

<u>ls</u>	50
<u>popd</u>	54
<u>pushd</u>	55
<u>pwd</u>	56
<u>vdir_lsearch</u>	57
<u>vname</u>	59
<u>what_is</u>	60
<u>what_is_list</u>	61
 <u>2</u>	
<u>General</u>	63
<u>? </u>	65
<u>alias</u>	66
<u>all_inputs</u>	67
<u>all_outputs</u>	68
<u>apropos</u>	69
<u>attribute_exists</u>	70
<u>clear</u>	71
<u>date</u>	72
<u>enable_transparent_latches</u>	73
<u>exec_embedded_script</u>	74
<u>exit</u>	76
<u>get_attribute</u>	77
<u>get_liberty_attribute</u>	79
<u>get_read_files</u>	80
<u>help</u>	81
<u>include</u>	82
<u>lcd</u>	83
<u>license</u>	84
<u>license_checkin</u>	85
<u>license_checkout</u>	86
<u>license_feature</u>	87
<u>license_list</u>	88
<u>license_version</u>	89
<u>lls</u>	90

Command Reference for Encounter RTL Compiler

<u>lpopd</u>	91
<u>lpushd</u>	92
<u>lpwd</u>	93
<u>man</u>	94
<u>more</u>	95
<u>quit</u>	97
<u>rc</u>	98
<u>redirect</u>	103
<u>reset_attribute</u>	105
<u>resume</u>	106
<u>sdc_shell</u>	107
<u>set_attribute</u>	108
<u>shell</u>	110
<u>stop_suspend</u>	111
<u>string_representation</u>	112
<u>suppress_messages</u>	113
<u>suspend</u>	114
<u>tcl_load</u>	115
<u>unsuppress_messages</u>	118
 <u>3</u>	
<u>GUI Text</u>	119
<u>General GUI Text Commands</u>	120
<u>gui_balloon_info</u>	121
<u>gui_hide</u>	121
<u>gui_info</u>	121
<u>gui_legend</u>	121
<u>gui_raise</u>	122
<u>gui_reset</u>	123
<u>gui_resume</u>	123
<u>gui_selection</u>	123
<u>gui_show</u>	124
<u>gui_status</u>	124
<u>gui_suspend</u>	124
<u>gui_update</u>	125

Command Reference for Encounter RTL Compiler

<u>HDL Viewer GUI Text Commands</u>	126
gui_hv_clear	127
gui_hv_get_file	127
gui_hv_load_file	127
gui_hv_set_indicators	128
<u>Schematic Viewer GUI Text Commands</u>	129
gui_sv_clear	130
gui_sv_cone	130
gui_sv_get_instance	130
gui_sv_grey	130
gui_sv_highlight	131
gui_sv_load	132
gui_sv_snapshot	133
gui_sv_toolbar_button	133
<u>Physical Viewer GUI Text Commands</u>	134
gui_pv_airline_add	136
gui_pv_airline_add_custom	137
gui_pv_airline_delete	138
gui_pv_airline_display	138
gui_pv_airline_raw_add	139
gui_pv_airline_raw_add_custom	140
gui_pv_clear	141
gui_pv_connectivity_airlines	141
gui_pv_deselect	141
gui_pv_display_collection	143
gui_pv_draw_box	144
gui_pv_draw_circle	145
gui_pv_draw_line	146
gui_pv_draw_triangle	147
gui_pv_highlight	148
gui_pv_highlight_update	150
gui_pv_label	150
gui_pv_preferences	151
gui_pv_redraw	151
gui_pv_select	151
gui_pv_selection	153

Command Reference for Encounter RTL Compiler

<u>gui_pv_snapshot</u>	153
<u>gui_pv_steiner_tree</u>	155
<u>gui_pv_update</u>	155
<u>gui_pv_zoom_box</u>	156
<u>gui_pv_zoom_fit</u>	157
<u>gui_pv_zoom_in</u>	157
<u>gui_pv_zoom_out</u>	157
<u>gui_pv_zoom_to</u>	157
4	
Chipware Developer	159
<u>cwd</u>	160
<u>cwd_check</u>	161
<u>cwd_create_check</u>	165
<u>cwd_report_check</u>	167
<u>hdl_create</u>	169
<u>hdl_create_binding</u>	170
<u>hdl_create_component</u>	172
<u>hdl_create_implementation</u>	174
<u>hdl_create_library</u>	176
<u>hdl_create_operator</u>	177
<u>hdl_create_package</u>	178
<u>hdl_create_parameter</u>	180
<u>hdl_create_pin</u>	182
5	
Input and Output	185
<u>decrypt</u>	188
<u>encrypt</u>	189
<u>export_critical_endpoints</u>	192
<u>read_cpf</u>	194
<u>read_db</u>	195
<u>read_def</u>	196
<u>read_dfm</u>	197
<u>read_dft_abstract_model</u>	199

Command Reference for Encounter RTL Compiler

<u>read_encounter</u>	200
<u>read_hdl</u>	201
<u>read_io_speclist</u>	205
<u>read_netlist</u>	206
<u>read_saif</u>	208
<u>read_sdc</u>	209
<u>read_spf</u>	211
<u>read_tcf</u>	212
<u>read_vcd</u>	213
<u>restore_design</u>	214
<u>split_db</u>	216
<u>write_atpg</u>	217
<u>write_bsdl</u>	218
<u>write_compression_macro</u>	219
<u>write_cpf</u>	220
<u>write_db</u>	221
<u>write_def</u>	223
<u>write_design</u>	224
<u>write_dft_abstract_model</u>	226
<u>write_do_ccd</u>	227
<u>write_do_ccd_compare_sdc</u>	228
<u>write_do_ccd_generate</u>	230
<u>write_do_ccd_propagate</u>	232
<u>write_do_ccd_validate</u>	234
<u>write_do_clp</u>	235
<u>write_do_lec</u>	238
<u>write_do_verify_cdc</u>	241
<u>write_encounter</u>	243
<u>write_et_atpg</u>	246
<u>write_et_bsv</u>	247
<u>write_et_dfa</u>	248
<u>write_et_lbist</u>	249
<u>write_et_mbist</u>	250
<u>write_et_rrfa</u>	251
<u>write_ets</u>	252
<u>write_ett</u>	253

Command Reference for Encounter RTL Compiler

<u>write_forward_saif</u>	254
<u>write_hdl</u>	255
<u>write_io_speclist</u>	259
<u>write_saif</u>	260
<u>write_scandef</u>	261
<u>write_script</u>	262
<u>write_sdc</u>	265
<u>write_sdf</u>	269
<u>write_set_load</u>	274
<u>write_spf</u>	275
<u>write_sv_wrapper</u>	276
<u>write_tcf</u>	279
<u>write_template</u>	280
6	
Constraints	283
<u>clock_uncertainty</u>	284
<u>create_mode</u>	287
<u>define_clock</u>	290
<u>define_cost_group</u>	295
<u>derive_environment</u>	296
<u>external_delay</u>	298
<u>generate_constraints</u>	302
<u>multi_cycle</u>	304
<u>path_adjust</u>	308
<u>path_delay</u>	312
<u>path_disable</u>	315
<u>path_group</u>	318
<u>propagate_constraints</u>	321
<u>specify_paths</u>	323
<u>validate_constraints</u>	329
7	
Elaboration and Synthesis	331
<u>elaborate</u>	332

<u>get_remove_assign_options</u>	335
<u>merge_to_multibit_cells</u>	336
<u>remove_assigns_without_optimization</u>	337
<u>remove_inserted_sync_enable_logic</u>	340
<u>retime</u>	341
<u>set_remove_assign_options</u>	344
<u>synthesize</u>	348
8	
<u>Analysis and Report</u>	353
<u>all_connected</u>	356
<u>all_des</u>	357
<u>all_des_inps</u>	358
<u>all_des_insts</u>	359
<u>all_des_outs</u>	360
<u>all_des_seqs</u>	361
<u>all_lib</u>	363
<u>all_lib_bufs</u>	364
<u>all_lib_ties</u>	365
<u>analyze_library_corners</u>	366
<u>check_design</u>	368
<u>clock_ports</u>	372
<u>compare_sdc</u>	373
<u>fanin</u>	374
<u>fanout</u>	377
<u>report</u>	380
<u>report_area</u>	385
<u>report_boundary_opto</u>	387
<u>report_cdn_loop_breaker</u>	389
<u>report_cell_delay_calculation</u>	390
<u>report_clock_gating</u>	391
<u>report_clocks</u>	397
<u>report_congestion</u>	399
<u>report_datapath</u>	400
<u>report_design_rules</u>	405

Command Reference for Encounter RTL Compiler

<u>report dft_chains</u>	406
<u>report dft_core_wrapper</u>	411
<u>report dft_registers</u>	415
<u>report dft_setup</u>	419
<u>report dft_violations</u>	423
<u>report disabled_transparent_latches</u>	426
<u>report gates</u>	427
<u>report hierarchy</u>	430
<u>report instance</u>	432
<u>report isolation</u>	435
<u>report level_shifter</u>	438
<u>report memory</u>	442
<u>report memory_cells</u>	443
<u>report messages</u>	444
<u>report multibit_inferencing</u>	446
<u>report net_cap_calculation</u>	449
<u>report net_delay_calculation</u>	450
<u>report net_res_calculation</u>	451
<u>report nets</u>	452
<u>report opcg_equivalents</u>	455
<u>report operand_isolation</u>	456
<u>report ple</u>	458
<u>report port</u>	460
<u>report power</u>	462
<u>report power_domain</u>	475
<u>report qor</u>	477
<u>report scan_compressibility</u>	481
<u>report sequential</u>	483
<u>report slew_calculation</u>	486
<u>report state_retention</u>	487
<u>report summary</u>	492
<u>report test_power</u>	494
<u>report timing</u>	498
<u>report units</u>	505
<u>report utilization</u>	506
<u>report yield</u>	507

Command Reference for Encounter RTL Compiler

<u>statistics</u>	508
<u>statistics add_metric</u>	509
<u>statistics log</u>	510
<u>statistics read</u>	512
<u>statistics remove_metric</u>	513
<u>statistics report</u>	514
<u>statistics reset</u>	517
<u>statistics run_stage_ids</u>	518
<u>statistics write</u>	519
<u>timestat</u>	520
<u>validate_timing</u>	521

9

<u>Physical</u>	523
<u>check_placement</u>	525
<u>create_group</u>	527
<u>create_placement_blockage</u>	528
<u>create_placement_halo_blockage</u>	529
<u>create_region</u>	530
<u>create_row</u>	532
<u>create_routing_blockage</u>	533
<u>create_routing_halo_blockage</u>	534
<u>create_track</u>	535
<u>def_move</u>	536
<u>generate_ple_model</u>	537
<u>generate_reports</u>	539
<u>move_blockage</u>	541
<u>move_instance</u>	542
<u>move_port</u>	543
<u>move_region</u>	544
<u>read_def</u>	545
<u>read_encounter</u>	549
<u>read_sdp_file</u>	550
<u>read_spf</u>	551
<u>report_congestion</u>	552

Command Reference for Encounter RTL Compiler

<u>report utilization</u>	553
<u>resize blockage</u>	554
<u>resize region</u>	555
<u>restore congestion map</u>	556
<u>save congestion map</u>	557
<u>specify floorplan</u>	558
<u>summary table</u>	560
<u>update congestion map</u>	562
<u>update gcell congestion</u>	563
<u>update gcell pin density</u>	564
<u>update gcell utilization</u>	565
<u>write_def</u>	566
<u>write_sdp_file</u>	567
<u>write_spf</u>	568

10

<u>Design for Test</u>	569
<u>add_opcg_hold_mux</u>	573
<u>analyze_scan_compressibility</u>	574
<u>analyze_testability</u>	583
<u>check_atpg_rules</u>	586
<u>check_dft_pad_configuration</u>	588
<u>check_dft_rules</u>	589
<u>check_mbist_rules</u>	595
<u>compress_block_level_chains</u>	598
<u>compress_scan_chains</u>	601
<u>concat_scan_chains</u>	615
<u>configure_pad_dft</u>	617
<u>connect_compression_clocks</u>	618
<u>connect_opcg_segments</u>	619
<u>connect_scan_chains</u>	620
<u>define_dft</u>	625
<u>define_dft_abstract_segment</u>	628
<u>define_dft_boundary_scan_segment</u>	633
<u>define_dft_dft_configuration_mode</u>	636

Command Reference for Encounter RTL Compiler

<u>define_dft_domain_macro_parameters</u>	639
<u>define_dft_fixed_segment</u>	641
<u>define_dft_floating_segment</u>	643
<u>define_dft_jtag_instruction</u>	645
<u>define_dft_jtag_instruction_register</u>	649
<u>define_dft_jtag_macro</u>	651
<u>define_dft_mbist_clock</u>	656
<u>define_dft_mbist_direct_access</u>	659
<u>define_dft_opcg_domain</u>	662
<u>define_dft_opcg_mode</u>	665
<u>define_dft_opcg_trigger</u>	667
<u>define_dft_osc_source</u>	669
<u>define_dft_preserved_segment</u>	671
<u>define_dft_scan_chain</u>	674
<u>define_dft_scan_clock_a</u>	680
<u>define_dft_scan_clock_b</u>	683
<u>define_dft_shift_enable</u>	686
<u>define_dft_shift_register_segment</u>	689
<u>define_dft_tap_port</u>	691
<u>define_dft_test_clock</u>	693
<u>define_dft_test_mode</u>	697
<u>dft_trace_back</u>	701
<u>fix_dft_violations</u>	703
<u>fix_scan_path_inversions</u>	707
<u>identify_multibit_cell_abstract_scan_segments</u>	708
<u>identify_shift_register_scan_segments</u>	710
<u>identify_test_mode_registers</u>	712
<u>insert_dft</u>	715
<u>insert_dft_boundary_scan</u>	717
<u>insert_dft_compression_logic</u>	721
<u>insert_dft_dfa_test_points</u>	732
<u>insert_dft_jtag_macro</u>	736
<u>insert_dft_lockup_element</u>	740
<u>insert_dft_logic_bist</u>	741
<u>insert_dft_mbist</u>	743
<u>insert_dft_opcg</u>	749

Command Reference for Encounter RTL Compiler

<u>insert_dft_ptam</u>	751
<u>insert_dft_rrfa_test_points</u>	754
<u>insert_dft_scan_power_gating</u>	760
<u>insert_dft_shadow_logic</u>	763
<u>insert_dft_test_point</u>	768
<u>insert_dft_user_test_point</u>	773
<u>insert_dft_wrapper_cell</u>	775
<u>insert_dft_wrapper_mode_decode_block</u>	780
<u>map_mbist_cgc_to_cgic</u>	782
<u>read_dft_abstract_model</u>	783
<u>read_io_speclist</u>	785
<u>replace_opcg_scan</u>	786
<u>replace_scan</u>	787
<u>report_dft_chains</u>	788
<u>report_dft_core_wrapper</u>	789
<u>report_dft_registers</u>	790
<u>report_dft_setup</u>	791
<u>report_dft_violations</u>	792
<u>report_scan_compressibility</u>	793
<u>report_test_power</u>	794
<u>reset_opcg_equivalent</u>	795
<u>reset_scan_equivalent</u>	796
<u>set_compatible_test_clocks</u>	797
<u>set_opcg_equivalent</u>	799
<u>set_scan_equivalent</u>	801
<u>update_scan_chains</u>	803
<u>write_atpg</u>	805
<u>write_bsdl</u>	808
<u>write_compression_macro</u>	811
<u>write_dft_abstract_model</u>	818
<u>write_dft_rtl_model</u>	821
<u>write_et_atpg</u>	822
<u>write_et_bsv</u>	830
<u>write_et_dfa</u>	833
<u>write_et_lbist</u>	837
<u>write_et_mbist</u>	840

Command Reference for Encounter RTL Compiler

<u>write_et_rrfa</u>	844
<u>write_io_speclist</u>	848
<u>write_logic_bist_macro</u>	850
<u>write_mbist_testbench</u>	852
<u>write_scandef</u>	856
11	
<u>Low Power Synthesis</u>	859
<u>build_rtl_power_models</u>	861
<u>clock_gating</u>	863
<u>clock_gating_connect_test</u>	865
<u>clock_gating_declone</u>	866
<u>clock_gating_import</u>	867
<u>clock_gating_insert_in_netlist</u>	869
<u>clock_gating_insert_obs</u>	870
<u>clock_gating_join</u>	872
<u>clock_gating_remove</u>	874
<u>clock_gating_share</u>	876
<u>clock_gating_split</u>	878
<u>read_saif</u>	880
<u>read_tcf</u>	885
<u>read_vcd</u>	890
<u>report_clock_gating</u>	893
<u>report_operand_isolation</u>	894
<u>report_power</u>	895
<u>state_retention</u>	896
<u>state_retention_connect_power_gating_pins</u>	897
<u>state_retention_swap</u>	898
<u>write_forward_saif</u>	899
<u>write_saif</u>	901
<u>write_tcf</u>	903
12	
<u>Advanced Low Power Synthesis</u>	905
<u>check_cpf</u>	906

<u>check_library</u>	908
<u>commit_cpf</u>	914
<u>create_library_domain</u>	915
<u>isolation_cell_remove</u>	916
<u>level_shifter_remove</u>	919
<u>read_cpf</u>	922
<u>reload_cpf</u>	924
<u>report_isolation</u>	925
<u>report_level_shifter</u>	926
<u>report_state_retention</u>	927
<u>verify_power_structure</u>	928
<u>write_cpf</u>	930
13	
Design Manipulation	933
<u>change_link</u>	935
<u>change_names</u>	937
<u>clock_gating</u>	944
<u>delete_unloaded_undriven</u>	945
<u>edit_netlist</u>	946
<u>edit_netlist_bitblast_all_ports</u>	948
<u>edit_netlist_bitblast_port</u>	949
<u>edit_netlist_connect</u>	950
<u>edit_netlist_dedicate_subdesign</u>	952
<u>edit_netlist_delete</u>	953
<u>edit_netlist_disconnect</u>	954
<u>edit_netlist_group</u>	956
<u>edit_netlist_hier_connect</u>	958
<u>edit_netlist_new_design</u>	959
<u>edit_netlist_new_instance</u>	960
<u>edit_netlist_new_port_bus</u>	962
<u>edit_netlist_new_primitive</u>	963
<u>edit_netlist_new_subport_bus</u>	965
<u>edit_netlist_ungroup</u>	966
<u>edit_netlist_uniquify</u>	967

Command Reference for Encounter RTL Compiler

<u>group</u>	968
<u>insert_tiehilo_cells</u>	969
<u>mv</u>	972
<u>remove_cdn_loop_breaker</u>	974
<u>reset_design</u>	976
<u>rm</u>	977
<u>ungroup</u>	978
<u>uniquify</u>	980
14	
<u>Customization</u>	981
<u>add_command_help</u>	982
<u>define_attribute</u>	983
<u>mesg_make</u>	987
<u>mesg_send</u>	989
<u>parse_options</u>	990
A	
<u>Applets</u>	995
<u>Introduction</u>	996
<u>applet</u>	997
<u>applet avail</u>	998
<u>applet install</u>	1000
<u>applet list</u>	1001
<u>applet load</u>	1002
<u>applet update</u>	1003
<u>applet version</u>	1004
<u>applet whatis</u>	1005
<u>Index</u>	1007

Alphabetical List of Commands

Symbols

? [65](#)

A

add_command_help [982](#)
add_opcg_hold_mux [573](#)
alias [66](#)
all_des_inps [358](#)
all_des_insts [359](#)
all_des_outs [360](#)
all_des_seqs [361](#)
all_lib [363](#)
all_lib_bufs [364](#)
all_lib_ties [365](#)
all_connected [356](#)
all_inputs [67](#)
all_outputs [68](#)
analyze_library_corners [366](#)
analyze_scan_compressibility [574](#)
analyze_testability [583](#)
applet [997](#)
applet_avail [998](#)
applet_install [1000](#)
applet_list [1001](#)
applet_load [1002](#)
applet_update [1003](#)
applet_version [1004](#)
applet_whatis [1005](#)
apropos [69](#)
attribute_exists [70](#)

B

basename [36](#)
build_rtl_power_models [861](#)

C

cd [37](#)
change_link [935](#)
change_names [937](#)

check_atpg_rules [586](#)
check_cpf [906](#)
check_design [368](#)
check_dft_pad_configuration [588](#)
check_dft_rules [589](#)
check_library [908](#)
check_mbist_rules [595](#)
check_placement [525](#)
clear [71](#)
clock_gating [863](#)
clock_gating_connect_test [865](#)
clock_gating_declone [866](#)
clock_gating_import [867](#)
clock_gating_insert_in_netlist [869](#)
clock_gating_insert_obs [870](#)
clock_gating_join [872](#)
clock_gating_remove [874](#)
clock_gating_share [876](#)
clock_gating_split [878](#)
clock_ports [372](#)
clock_uncertainty [284](#)
commit_cpf [914](#)
compare_sdc [373](#)
compress_block_level_chains [598](#)
compress_scan_chains [601](#)
concat_scan_chains [615](#)
configure_pad_dft [617](#)
connect_compression_clocks [618](#)
connect_opcg_segments [619](#)
connect_scan_chains [620](#)
create_group [527](#)
create_library_domain [915](#)
create_mode [287](#)
create_placement_blockage [528](#)
create_placement_halo_blockage [529](#)
create_region [530](#)
create_routing_blockage [533](#)
create_row [532](#)
create_track [535](#)
cwd [160](#)
cwd_check [161](#)
cwd_create_check [165](#)
cwd_report_check [167](#)

D

date [72](#)
decrypt [188](#)
define_attribute [983](#)
define_clock [290](#)
define_cost_group [295](#)
define_dft [625](#)
define_dft abstract_segment [628](#)
define_dft boundary_scan_segment [633](#)
define_dft dft_configuration_mode [636](#)
define_dft domain_macro_parameters [639](#)
define_dft fixed_segment [641](#)
define_dft floating_segment [643](#)
define_dft jtag_instruction [645](#)
define_dft jtag_instruction_register [649](#)
define_dft jtag_macro [651](#)
define_dft mbist_clock [656](#)
define_dft mbist_direct_access [659](#)
define_dft opcg_domain [662](#)
define_dft opcg_trigger [667](#)
define_dft osc_source [669](#)
define_dft preserved_segment [671](#)
define_dft scan_chain [674](#)
define_dft scan_clock_a [680](#)
define_dft scan_clock_b [683](#)
define_dft shift_enable [686](#)
define_dft shift_register_segment [689](#)
define_dft tap_port [691](#)
define_dft test_clock [656, 693](#)
define_dft test_mode [697](#)
delete_unloaded_undriven [945](#)
derive_environment [296](#)
dft_trace_back [701](#)
dirname [39](#)
dirs [40](#)

E

edit_netlist [946](#)
edit_netlist bitblast_all_ports [948](#)
edit_netlist bitblast_port [949](#)
edit_netlist connect [950](#)
edit_netlist dedicate_subdesign [952](#)
edit_netlist disconnect [954](#)
edit_netlist group [956, 968](#)
edit_netlist hier_connect [958](#)
edit_netlist new_design [959](#)
edit_netlist new_instance [960](#)

edit_netlist new_port_bus [962](#)
edit_netlist new_primitive [963](#)
edit_netlist new_support_bus [965](#)
edit_netlist ungroup [966](#)
edit_netlist unify [967, 980](#)
elaborate [332](#)
enable_transparent_latches [73](#)
encrypt [189](#)
exec_embedded_script [74](#)
exit [76](#)
export_critical_endpoints [192](#)
external_delay [298](#)

F

fanin [374](#)
fanout [377](#)
filter [41](#)
find [44](#)
fix_dft_violations [703](#)
fix_scan_path_inversions [707](#)

G

generate_constraints [302](#)
generate_ple_model [537](#)
generate_reports [539](#)
get_attribute [77](#)
get_liberty_attribute [79](#)
get_read_files [80](#)
get_remove_assign_options [335](#)
group [968](#)
gui_balloon_info [121](#)
gui_hide [121](#)
gui_hv_clear [127](#)
gui_hv_get_file [127](#)
gui_hv_load_file [127](#)
gui_hv_set_indicators [128](#)
gui_info [121](#)
gui_legend [121](#)
gui_pv_airline_add [136](#)
gui_pv_airline_add_custom [137](#)
gui_pv_airline_delete [138](#)
gui_pv_airline_display [138](#)
gui_pv_airline_raw_add [139](#)
gui_pv_airline_raw_add_custom [140](#)
gui_pv_clear [141](#)
gui_pv_connectivity_airlines [141](#)
gui_pv_deselect [141](#)

Command Reference for Encounter RTL Compiler

gui_pv_display_collection [143](#)
gui_pv_draw_box [144](#)
gui_pv_draw_circle [145](#), [147](#)
gui_pv_draw_line [146](#)
gui_pv_highlight [148](#)
gui_pv_highlight_update [150](#)
gui_pv_label [150](#)
gui_pv_preferences [151](#)
gui_pv_redraw [151](#)
gui_pv_select [151](#)
gui_pv_selection [153](#)
gui_pv_snapshot [153](#)
gui_pv_steiner_tree [155](#)
gui_pv_update [155](#)
gui_pv_zoom_box [156](#)
gui_pv_zoom_fit [157](#)
gui_pv_zoom_in [157](#)
gui_pv_zoom_out [157](#)
gui_pv_zoom_to [157](#)
gui_raise [122](#)
gui_reset [123](#)
gui_selection [123](#)
gui_show [124](#)
gui_status [124](#)
gui_sv_clear [130](#)
gui_sv_cone [130](#)
gui_sv_get_instance [130](#)
gui_sv_grey [130](#)
gui_sv_highlight [131](#)
gui_sv_load [132](#)
gui_sv_snapshot [133](#)
gui_sv_toolbar_button [133](#)
gui_update [125](#)

H

hdl_create [169](#)
hdl_create binding [170](#)
hdl_create component [172](#)
hdl_create implementation [174](#)
hdl_create library [176](#)
hdl_create operator [177](#)
hdl_create package [178](#)
hdl_create parameter [180](#)
hdl_create pin [182](#)
help [81](#)

I

identify_multibit_cell_abstract_scan_segments [708](#)
identify_shift_register_scan_segments [710](#)
identify_test_mode_registers [712](#)
include [82](#)
inout_mate [48](#)
insert_dft [715](#)
insert_dft_boundary_scan [717](#)
insert_dft_compression_logic [721](#)
insert_dft_jtag_macro [736](#)
insert_dft_lockup_element [740](#)
insert_dft_logic_bist [741](#)
insert_dft_mbist [743](#)
insert_dft_opcg [749](#)
insert_dft_ptam [751](#)
insert_dft_rrfa_test_points [754](#)
insert_dft_scan_power_gating [760](#)
insert_dft_shadow_logic [763](#)
insert_dft_test_point [768](#)
insert_dft_user_test_point [773](#)
insert_dft_wrapper_cell [775](#)
insert_dft_wrapper_mode_decode_block [780](#)
insert_tiehilo_cells [969](#)
isolation_cell_remove [916](#)

L

lcd [83](#)
level_shifter_remove [919](#)
license [84](#)
license_checkin [85](#)
license_checkout [86](#)
license_feature [87](#)
license_list [88](#)
license_version [89](#)
ll [49](#)
lIs [90](#)
lpopd [91](#)
lpushd [92](#)
lpwd [93](#)
ls [50](#)

M

man [94](#)
map_mbist_cgc_to_cgic [782](#)
merge_to_multibit_cells [336](#)
mesg_make [987](#)
mesg_send [989](#)
more [95](#)
move_blockage [541](#)
move_instance [542](#)
move_port [543](#)
move_region [544](#)
multi_cycle [304](#)
mv [972](#)

read_tcf [885](#)
read_vcd [890](#)
redirect [103](#)
reload_cpf [924](#)
remove_assigns_without_optimization [33](#)
[7](#)
remove_cdn_loop_breaker [974](#)
remove_inserted_sync_enable_logic [340](#)
replace_opcg_scan [786](#)
replace_scan [787](#)
report [380](#)
report_area [385](#)
report_boundary_opto [387](#)
report_cdn_loop_breaker [389](#)
report_cell_delay_calculation [390](#)
report_clock_gating [391](#)
report_clocks [397](#)
report_congestion [399](#)
report_datapath [400](#)
report_design_rules [405](#)
report_dft_chains [406](#)
report_dft_core_wrapper [411](#)
report_dft_registers [415](#)
report_dft_setup [419](#)
report_dftViolations [423](#)
report_disabled_transparent_latches [426](#)
report_gates [427](#)
report_hierarchy [430](#)
report_instance [432](#)
report_isolation [435](#)
report_level_shifter [438](#)
report_memory [442](#)
report_memory_cells [443](#)
report_messages [444](#)

report_multibit_inferencing [446](#)
report_net_cap_calculation [449](#)
report_net_delay_calculation [450](#)
report_net_res_calculation [451](#)
report_nets [452](#)
report_opcg_equivalents [455](#)
report_operand_isolation [456](#)
report_ple [458](#)
report_port [460](#)
report_power [462](#)
report_power_domain [475](#)
report_qor [477](#)
report_scan_compressibility [481](#)
report_sequential [483](#)
report_slew_calculation [486](#)
report_state_retention [487](#)
report_summary [492](#)

P

parse_options [990](#)
path_adjust [308](#)
path_delay [312](#)
path_disable [315](#)
path_group [318](#)
popd [54](#)
propagate_constraints [321](#)
pushd [55](#)
pwd [56](#)

Q

quit [97](#)

R

rc [98](#)
read_cpf [922](#)
read_db [195](#)
read_def [545](#)
read_dfm [197](#)
read_dft_abstract_model [783](#)
read_encounter [549](#)
read_hdl [201](#)
read_io_speclist [785](#)
read_netlist [206](#)
read_saif [880](#)
read_sdc [209](#)
read_sdp_file [550](#)
read_spf [551](#)

report_multibit_inferencing [446](#)
report_net_cap_calculation [449](#)
report_net_delay_calculation [450](#)
report_net_res_calculation [451](#)
report_nets [452](#)
report_opcg_equivalents [455](#)
report_operand_isolation [456](#)
report_ple [458](#)
report_port [460](#)
report_power [462](#)
report_power_domain [475](#)
report_qor [477](#)
report_scan_compressibility [481](#)
report_sequential [483](#)
report_slew_calculation [486](#)
report_state_retention [487](#)
report_summary [492](#)

Command Reference for Encounter RTL Compiler

report test_power [494](#)
report timing [498](#)
report units [505](#)
report yield [507](#)
reset_attribute [105](#)
reset_design [976](#)
reset_opcg_equivalent [795](#)
reset_scan_equivalent [796](#)
resize_blockage [554](#)
resize_region [555](#)
restore_congestion_map [556](#)
restore_design [214](#)
resume [106](#)
retime [341](#)
rm [953, 977](#)

S

save_congestion_map [557](#)
sdc_shell [107](#)
set_attribute [108](#)
set_compatible_test_clocks [797](#)
set_opcg_equivalent [799](#)
set_remove_assign_options [344](#)
set_scan_equivalent [801](#)
shell [110](#)
specify_floorplan [558](#)
specify_paths [323](#)
split_db [216](#)
state_retention [896](#)
state_retention
 connect_power_gating_pins [897](#)
state_retention_swap [898](#)
statistics [508](#)
statistics add_metric [509](#)
statistics log [510](#)
statistics read [512](#)
statistics remove_metric [513](#)
statistics report [514](#)
statistics reset [517](#)
statistics run_stage_ids [518](#)
statistics write [519](#)
stop_suspend [111](#)
string_representation [112](#)
summary_table [560](#)
suppress_messages [113](#)
suspend [114](#)
synthesize [348](#)

T

timestat [520](#)

U

ungroup [978](#)
unsuppress_messages [118](#)
update_congestion_map [562](#)
update_gcell_congestion [563](#)
update_gcell_pin_density [564](#)
update_gcell_utilization [565](#)
update_scan_chains [803](#)

V

validate_constraints [329](#)
validate_timing [521](#)
vdir_lsearch [57](#)
verify_power_structure [928](#)
vname [59](#)

W

what_is [60](#)
what_is_list [61](#)
write_atpg [805](#)
write_bsdl [808](#)
write_compression_macro [811](#)
write_cpf [930](#)
write_db [221](#)
write_def [566](#)
write_design [224](#)
write_dft_abstract_model [818](#)
write_dft_rtl_model [821](#)
write_do_ccd [227](#)
write_do_ccd_compare_sdc [228](#)
write_do_ccd_generate [230](#)
write_do_ccd_validate [234](#)
write_do_clp [235](#)
write_do_lec [238](#)
write_do_verify_cdc [241](#)
write_encounter [243](#)
write_et_atpg [822, 823, 824, 844](#)
write_et_bsv [830](#)
write_et_dfa [833](#)
write_et_lbist [837](#)

Command Reference for Encounter RTL Compiler

write_et_mbist [840](#)
write_ets [252](#)
write_ett [253](#)
write_forward_saif [899](#)
write_hdl [255](#)
write_io_speclist [848](#)
write_logic_bist_macro [850](#)
write_mbist_testbench [852](#)
write_saif [901](#)
write_scandef [856](#)
write_script [262](#)
write_sdc [265](#)
write_sdf [269](#)
write_sdp_file [567](#)
write_set_load [274](#)
write_spf [568](#)
write_sv_wrapper [276](#)
write_tcf [903](#)
write_template [280](#)

Preface

- [About This Manual](#) on page 26
- [Additional References](#) on page 26
- [How to Use the Documentation Set](#) on page 27
- [Reporting Problems or Errors in Manuals](#) on page 28
- [Customer Support](#) on page 29
- [Messages](#) on page 30
- [Man Pages](#) on page 31
- [Command-Line Help](#) on page 32
- [Documentation Conventions](#) on page 34

About This Manual

This manual provides a concise reference of the commands available to the user when using Encounter™ RTL Compiler. This manual describes each command available within the RTL Compiler shell with their command options.

Additional References

The following sources are helpful references, but are not included with the product documentation:

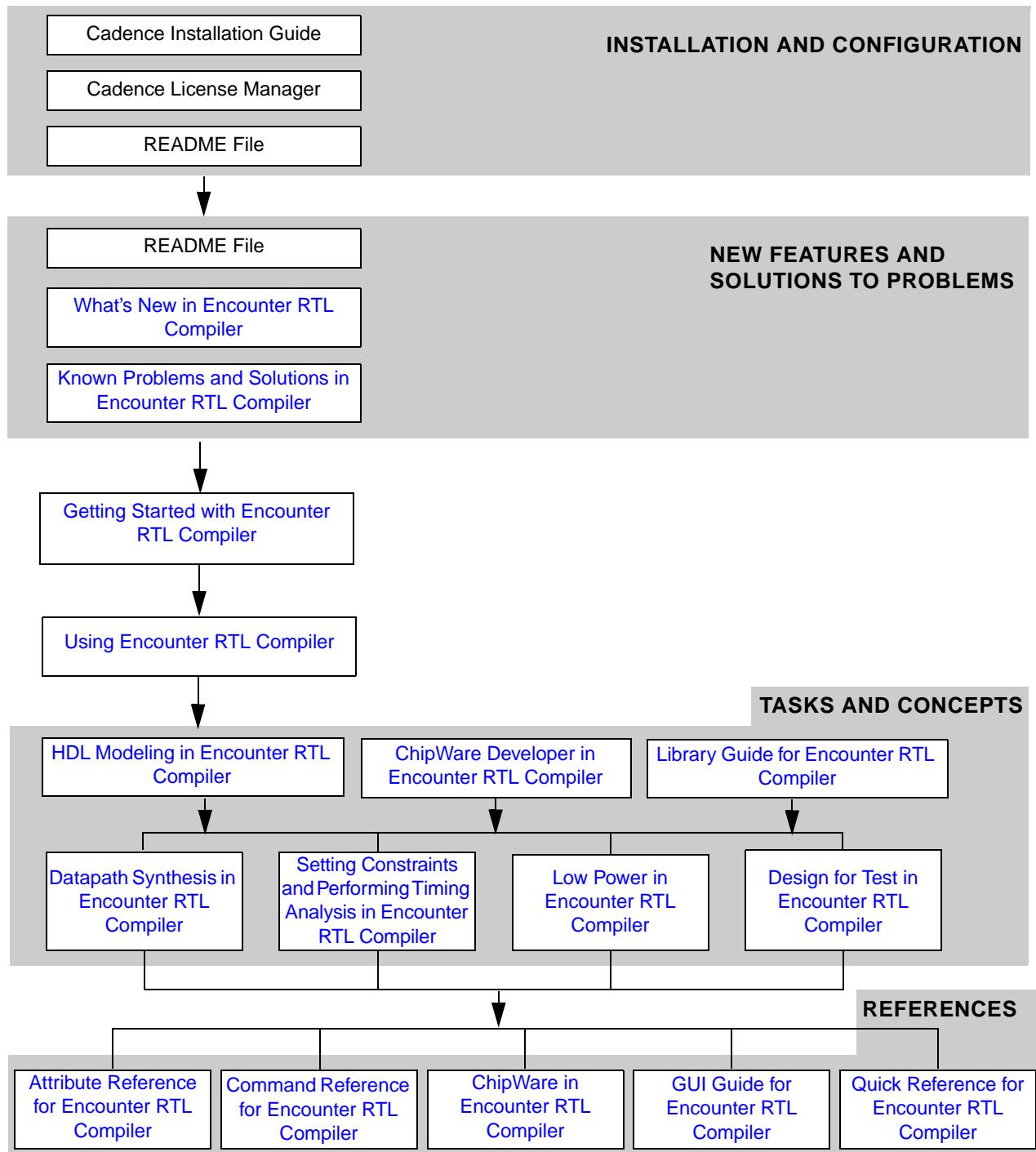
- TclTutor, a computer aided instruction package for learning the Tcl language:
<http://www.msen.com/~clif/TclTutor.html>.
- TCL Reference, *Tcl and the Tk Toolkit*, John K. Ousterhout, Addison-Wesley Publishing Company
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std.1364-1995)
- IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std. 1364-2001)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1987)
- IEEE Standard VHDL Language Reference Manual (IEEE Std. 1076-1993)

Note: For information on purchasing IEEE specifications go to <http://shop.ieee.org/store/> and click on *Standards*.

Command Reference for Encounter RTL Compiler

Preface

How to Use the Documentation Set



Reporting Problems or Errors in Manuals

The Cadence® Help online documentation, lets you view, search, and print Cadence product documentation. You can access Cadence Help by typing `cdnshelp` from your Cadence tools hierarchy.

Contact Cadence Customer Support to file a CCR if you find:

- An error in the manual
- An omission of information in a manual
- A problem using the Cadence Help documentation system

Customer Support

Cadence offers live and online support, as well as customer education and training programs.

Cadence Online Support

The Cadence® online support website offers answers to your most common technical questions. It lets you search more than 40,000 FAQs, notifications, software updates, and technical solutions documents that give you step-by-step instructions on how to solve known problems. It also gives you product-specific e-mail notifications, software updates, service request tracking, up-to-date release information, full site search capabilities, software update ordering, and much more.

For more information on Cadence online support go to:

<http://support.cadence.com>

Other Support Offerings

- **Support centers**—Provide live customer support from Cadence experts who can answer many questions related to products and platforms.
- **Software downloads**—Provide you with the latest versions of Cadence products.
- **Education services**—Offers instructor-led classes, self-paced Internet, and virtual classroom.
- **University software program support**—Provides you with the latest information to answer your technical questions.

For more information on these support offerings go to:

<http://www.cadence.com/support>

Messages

From within RTL Compiler there are two ways to get information about error messages.

- Use the `report messages` command.

For example:

```
rc:/> report messages
```

This returns the detailed information for each message output in your current RTL Compiler run. It also includes a summary of how many times each message was issued.

- Use the `man` command.

Note: You can only use the `man` command for messages within RTL Compiler.

For example, to get more information about the "TIM-11" message, type the following command:

```
rc:/> man TIM-11
```

If you do not get the details that you need or do not understand a message, either contact Cadence Customer Support to file a PCR or email the message ID you would like improved to:

`rc_msg_improvement@cadence.com`

Man Pages

In addition to the Command and Attribute References, you can also access information about the commands and attributes using the man pages in RTL Compiler. Man pages contain the same content as the Command and Attribute References.

To use our man pages from the UNIX shell:

1. Set your environment to view the correct directory:

```
setenv MANPATH $CDN_SYNTH_ROOT/share/synth/man
```

2. Enter the name of the command or attribute that you want. For example:

- man check_dft_rules
- man cell_leakage_power

You can also use the `more` command, which behaves like its UNIX counterpart. If the output of a manpage is too small to be displayed completely on the screen, use the `more` command to break up the output. Use the spacebar to page forward, like the UNIX `more` command.

```
rc:/> more man synthesize
```

Command-Line Help

You can get quick syntax help for commands and attributes at the RTL Compiler command-line prompt. There are also enhanced search capabilities so you can more easily search for the command or attribute that you need.

Note: The command syntax representation in this document does not necessarily match the information that you get when you type `help command_name`. In many cases, the order of the arguments is different. Furthermore, the syntax in this document includes all of the dependencies, where the help information does this only to a certain degree.

If you have any suggestions for improving the command-line help, please e-mail them to:

`rc_pubs@cadence.com`

Getting the Syntax for a Command

Type the `help` command followed by the command name.

For example:

`rc:/> help path_delay`

This returns the syntax for the `path_delay` command.

Getting the Syntax for an Attribute

Type the following:

`rc:/> get_attribute attribute_name * -help`

For example:

`rc:/> get_attribute max_transition * -help`

This returns the syntax for the `max_transition` attribute.

Searching for Attributes

You can get a list of all the available attributes by typing the following command:

```
rc:/> get_attribute * * -h
```

You can type a sequence of letters after the `set_attribute` command and press Tab to get a list of all attributes that contain those letters.

```
rc:/> set_attr li
ambiguous "li": lib lef consistency_check_enable lib_search_path libcell
liberty_attributes libpin library library_domain line_number
```

Searching For Commands When You Are Unsure of the Name

You can use help to find a command if you only know part of its name, even as little as one letter.

- You can type a single letter and press Tab to get a list of all commands that start with that letter.

For example:

```
rc:/> c <Tab>
```

This returns the following commands:

```
ambiguous "c": cache_vname calling_proc case catch cd cdsdoc change_names
check_dft_rules chipware clear clock clock_gating clock_ports close cmdExpand
command_is_complete concat configure_pad_dft connect_scan_chains continue
cwd_install ..
```

- You can type a sequence of letters and press Tab to get a list of all commands that start with those letters.

For example:

```
rc:/> path_<Tab>
```

This returns the following commands:

```
ambiguous command name "path_": path_adjust path_delay path_disable path_group
```

Documentation Conventions

To aid the readers understanding a consistent formatting style has been used throughout this manual.

- UNIX commands are shown following the `unix>` string.
- RTL Compiler commands are shown following the `rc:/>` string.

Text Command Syntax

The list below describes the syntax conventions used for the RTL Compiler text commands.

<code>literal</code>	Nonitalic words indicate keywords that you must type literally. These keywords represent command, attribute or option names
<code>arguments and options</code>	Words in italics indicate user-defined arguments or options for which you must substitute a name or a value.
<code> </code>	Vertical bars (OR-bars) separate possible choices for a single argument.
<code>[]</code>	Brackets denote options. When used with OR-bars, they enclose a list of choices from which you can choose one.
<code>{ }</code>	Braces denote arguments and are used to indicate that a choice is required from the list of arguments separated by OR-bars. You must choose one from the list <code>{ argument1 argument2 argument3 }</code>
<code>{ }</code>	Braces in bold-face type must be entered literally.
	Braces, used in Tcl command examples, indicate that the braces must be typed in.
<code>...</code>	Three dots (...) indicate that you can repeat the previous argument. If the three dots are used with brackets (that is, <code>[argument]...</code>), you can specify zero or more arguments. If the three dots are used without brackets (<code>argument...</code>), you must specify at least one argument, but can specify more.
<code>#</code>	The pound sign precedes comments in command files.

Navigation

- [basename](#) on page 36
- [cd](#) on page 37
- [dirname](#) on page 39
- [dirs](#) on page 40
- [filter](#) on page 41
- [find](#) on page 44
- [inout_mate](#) on page 48
- [ll](#) on page 49
- [ls](#) on page 50
- [popd](#) on page 54
- [pushd](#) on page 55
- [pwd](#) on page 56
- [vdir_lsearch](#) on page 57
- [vname](#) on page 59
- [what_is](#) on page 60
- [what_is_list](#) on page 61

basename

`basename pathname`

Removes the leading directory names of the specified path name and returns only the object name. This command behaves similarly to the UNIX basename command.

Options and Arguments

<i>pathname</i>	Specifies the path name of the object, including the object name.
-----------------	---

Examples

- The following example removes the directory name of the CW_absval ChipWare component and only returns the component name:

```
rc:/> basename \
      /hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval
      CW_absval
```

- The following example uses the basename command with the dirname command to return the name of the library to which the ND2X1 library cell belongs:

```
rc:/> set libcell /libraries/LIB/libcells/ND2X1
rc:/> basename [dirname [dirname $libcell]]
```

Related Information

Related commands: [dirname on page 39](#)

Command Reference for Encounter RTL Compiler

Navigation

cd

`cd [directory]`

Sets the current directory in the design hierarchy and navigates the design hierarchy. This command is similar to its UNIX counterpart. A description of the design hierarchy is given in the [Using Encounter RTL Compiler](#).

Options and Arguments

<i>directory</i>	Specifies the name of the directory to be set as the current directory. The “.”, “..”, and “/” have the same meaning as their UNIX counterparts (current directory, parent directory, and root directory respectively). You can use wildcards when they do not produce an ambiguous reference (more than one match).
------------------	--

Examples

- The following command returns you to the top of the design hierarchy:

```
rc:/designs> cd  
rc:/> pwd  
/
```

- The following command specifies the absolute path to the target directory:

```
rc:/> cd /designs/alu/timing/exceptions
```

- The following command specifies the relative path to the target directory:

```
rc:/> cd designs/alu
rc:/designs/alu> cd timing/exceptions
```

- The following command changes the current directory to a parent directory:

```
rc:/designs/alu/timing/exceptions cd ../../subdesigns  
rc:/designs/alu/subdesigns> pwd  
/designs/alu/subdesigns
```

- The following command uses wildcards in the path specification:

```
rc:/designs/cmplx_alu/subdesigns/addinc> cd ../../tim*/exc*
rc:/designs/cmplx_alu/timing/exceptions> pwd
/designs/cmplx_alu/timing/exceptions
```

Command Reference for Encounter RTL Compiler

Navigation

Related Information

Affects these commands:

[dirs](#) on page 40
[ls](#) on page 50
[popd](#) on page 54
[pushd](#) on page 55
[pwd](#) on page 56

Command Reference for Encounter RTL Compiler

Navigation

dirname

`dirname pathname [-times integer]`

Removes the object name of the specified path name and only returns the directory name. This command behaves similarly to the UNIX dirname command.

Options and Arguments

<i>pathname</i>	Specifies the path name of the object, including the object name.
-times <i>integer</i>	Specifies the number of times to apply dirname. <i>Default:</i> 1

Examples

- The following example removes the `CW_absval` ChipWare component name and only returns its directory name:

```
rc:/>dirname \
    /hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval
/hdl_libraries/CW/components/CW_absval/comp_architectures
```

- The following command applies the `dirname` command 3 times to the specified path.

```
rc:/> dirname -times 3 \ /hdl_libraries/CW/components/CW_absval/comp_architectures/CW_absval  
/hdl_libraries/CW/components
```

- The following example uses the basename command with the dirname command to return the name of the library to which the `ND2X1` library cell belongs:

```
rc:/> set libcell /libraries/LIB/libcells/ND2X1  
rc:/> basename [dirname [dirname $libcell]]
```

Related Information

Related commands: [basename](#) on page 36

dirs

dirs

Displays the contents of the design directory stack. This command is similar to its UNIX counterpart and is used in conjunction with the [pushd](#) and [popd](#) commands.

Example

- The following commands respectively add designs and libraries to the design directory stack, then display the contents of the directory stack:

```
rc:/> pushd designs  
/designs /  
rc:/designs> pushd /libraries  
/libraries /designs /  
rc:/libraries> dirs  
/libraries /designs /
```

- The following commands respectively remove the last added directories and then display the contents of the directory stack:

```
rc:/libraries> popd  
/designs /  
rc:/designs> popd  
/  
rc:/> dirs  
/
```

Related Information

Related commands:	<u>ls</u> on page 50 <u>popd</u> on page 54 <u>pushd</u> on page 55 <u>pwd</u> on page 56
-------------------	--

filter

```
filter [-invert] [-special] [-vname]
       [-expr string] [-regexp]
       attribute_name attribute_value [object_list]
```

Filters a set of objects based on the values of the given attributes using the pattern matching mechanism from the `glob` Tcl command.

This command provides a powerful means of selecting objects within the design hierarchy at a more discrete level than is allowed by the directory structure alone.

Use the `get_attribute` command to list the attributes available for each object type.

Options and Arguments

<i>attribute_name</i>	Specifies the name of an attribute to use as filter. This argument is required. A compound string (containing spaces) should be represented as a list either by using double-quotes or braces ({ }).
<i>attribute_value</i>	Specifies the value of an attribute to use as filter.
<code>-expr</code> <i>string</i>	Specifies an expression that can be used to compare attribute values. Applies to numerical expressions only.
<code>-invert</code>	Filters out objects that match the expression and returns those that do not.
<i>object_list</i>	Specifies a Tcl list of objects to filter.
<code>-regexp</code>	Overrides the default Tcl glob pattern matching with Tcl regular expression matching.
<code>-special</code>	Use this option when the attribute value contains special characters, such as square brackets.
<code>-vname</code>	Allows to specify the Verilog name instead of the RTL Compiler design hierarchy path name

Examples

- The following command finds the list of all matching library cells on which the `preserve` attribute has been set to true:

```
rc:/> filter preserve true [find . -libcell *]
```

Command Reference for Encounter RTL Compiler

Navigation

- The following command stores the Tcl list of all matching cells returned by the `filter` command in a variable for use in scripting later.

```
rc:/> set preserved_cells [filter preserve true [find . -libcell *]]
```

- The following command embeds the Tcl list of all matching cells returned by the `filter` command as part of a larger command.

```
rc:/> report timing -through [filter preserve true [find . -libcell *]]
```

- The following Tcl code fragment sets the variable `result` to all instances that start with the letter `g` and whose corresponding library cell starts with `inv`:

```
set result {}
foreach inst [find . -inst g*] {
    if {[string match "inv*" [get_att libcell $inst]]}
        {lappend result $inst}
}
puts $result
```

- The following example returns only returns those pins that have the `preserve` attribute set to either `true` or `false` and ignores those with `size_ok` values:

```
rc:/> filter -regexp preserve {true|false} [find / -pin *]
```

- The following command finds all the instances of cell `bufx1` in library `mylib` in design `dut_shell`:

```
filter -regexp libcell {.*bufx1} [find /designs/dut_shell -instance *]
```

- Use the `ls -dir` command to format the output:

```
rc:/> ls -dir [filter preserve true [find . -libcell *]]
/libraries/penny/libcells/ANTENNA/
/libraries/penny/libcells/FILL1/
/libraries/penny/libcells/RF1R1WX2/
/libraries/penny/libcells/RF2R1WX2/
```

- Use the `ls -dir` command and the redirect arrow to redirect the output to the specified file:

```
rc:/> ls -dir [filter preserve true [find . -libcell *]] > filter.txt
```

You can also append arrows ("`>>`").

- Use the `-special` option to handle the special characters in the attribute value.

```
rc:/> set instances [find /designs/* -instance instances_seq/*]
{/designs/test/instances_seq/out_reg[3]}{/designs/test/instances_seq/out_
reg[2]}{/designs/test/instances_seq/out_reg[0]}{/designs/test/instances_
seq/out_reg[1]}
rc:/> filter -special dft_test_clock \
{/designs/test/dft/test_clock_domainsclk[0]/clk[0]} $instances
{/designs/test/instances_seq/out_reg[0]}{/designs/test/instances_seq/out_
reg[1]}
```

Command Reference for Encounter RTL Compiler

Navigation

- The following command returns the design whose tns value equals 0.

```
rc:/> get_attr tns [find / -design mydesign]  
0  
rc:/> filter -expr == tns 0 [find / -design *]  
/designs/mydesign
```

- The following command returns the design whose tns value is larger than -1.

```
rc:/> filter -expr > tns -1 [find / -design *]  
/designs/mydesign
```

Related Information

Affects these commands:

[ls](#) on page 50

[get_attribute](#) on page 77

[set_attribute](#) on page 108

find

```
find [root_path]
      [-maxdepth integer] [-mindepth integer]
      [-ignorecase] [-regexp <expression>]
      [-split] [-vname] {-option...|-*} object
```

Searches the design hierarchy for the specified types of objects and returns a Tcl list containing the full paths to any matching objects. This list can then be used by other commands to operate on groups of objects.

The `find` command supports the * and ? wildcard characters.

Note: The `find` command is very powerful but overusing it can increase the execution time. To reduce the execution time, be as specific as possible when specifying the object to match.

Options and Arguments

<code>-ignorecase</code>	Ignores the case (upper case or lower case) of the parameters. Alternatively, specifies the search to be case insensitive.
<code>-maxdepth level</code>	Descends no more than the specified number (non-negative integer) of levels below <code>root_path</code> . A level of 0 searches only the <code>root_path</code> . <i>Default:</i> infinity
<code>-mindepth integer</code>	Skips the specified number (non-negative integer) of levels below <code>root_path</code> before finding objects. A level of 1 searches all objects except <code>root_path</code> . <i>Default:</i> 0
<code>object</code>	Specifies the name of the object to match. The name can include wildcard characters.
<code>-option</code>	Specifies the type of object you want to find. You can specify multiple options or look in all types by specifying *. Check the command help for a list of the valid object types.
<code>-regexp</code>	Specifies to find the specified regular expression.
<code>root_path</code>	Specifies the name of the directory from where to start searching. The name can include wildcard characters.

Command Reference for Encounter RTL Compiler

Navigation

By default, an explicit search is done for the specified object. During an explicit search every object directory is searched, instead of starting from *root_path*. Specifying a *root_path* reduces the numbers of locations where the `find` command will search which reduces the execution time.

`-split`

Causes the command to return one object per line.

This option is for interactive use only: the Tcl result is set to null and the objects are only printed to the screen.

`-vname`

Removes the container directories in the path name and returns Verilog style names where appropriate. This option will only work on the following objects: pin, port, net, subdesign, and instance.

Examples

- The following command searches for any object type whose name contains `add`, starting from the current directory (`/designs`):

```
rc:/designs> find . * *add*
/designs/alu/instances_hier/ops1_add_25 /designs/alu/subdesigns/addinc64
```

- The following command finds all registers in all designs:

```
rc:/> find des* -instance *seq/*
/designs/alu/instances_hier/RC_CG_HIER_INST_0/instances_seq/RC_CGIC_INST
/designs/alu/instances_seq/aluout_reg_7 /designs/alu/instances_seq/aluout_
reg_6 /designs/alu/instances_seq/aluout_reg_4 /designs/alu/instances_
seq/aluout_reg_0 /designs/alu/instances_seq/aluout_reg_3
/designs/alu/instances_seq/aluout_reg_5 /designs/alu/instances_seq/aluout_
reg_2 /designs/alu/instances_seq/aluout_reg_1 /designs/alu/instances_
seq/zero_reg
```

- The following command finds all input ports in design `alu`:

```
rc:/> find des*/alu -port ports_in/*
{/designs/alu/ports_in/opcode[2]}{/designs/alu/ports_in/opcode[1]}
{/designs/alu/ports_in/opcode[0]}{/designs/alu/ports_in/data[7]}
{/designs/alu/ports_in/data[6]}{/designs/alu/ports_in/data[5]}
{/designs/alu/ports_in/data[4]}{/designs/alu/ports_in/data[3]}
{/designs/alu/ports_in/data[2]}{/designs/alu/ports_in/data[1]}
{/designs/alu/ports_in/data[0]}{/designs/alu/ports_in/accum[7]}
{/designs/alu/ports_in/accum[6]}{/designs/alu/ports_in/accum[5]}
{/designs/alu/ports_in/accum[4]}{/designs/alu/ports_in/accum[3]}
{/designs/alu/ports_in/accum[2]}{/designs/alu/ports_in/accum[1]}
{/designs/alu/ports_in/accum[0]}/designs/alu/ports_in/clock
/designs/alu/ports_in/ena /designs/alu/ports_in/reset
```

Command Reference for Encounter RTL Compiler

Navigation

- The following command searches for an external delay whose name starts with `in`, starting from the `designs` directory:

```
rc:/designs> find designs -external_delay in*
/designs/alu/timing/external_delays/in_del_1
```

- The following command finds all designs that are four characters:

```
rc:/> find . -designs ****
/designs/test
```

- The following command finds all design names with four characters that end with the letter "i":

```
rc:/> find . -designs ???i
/designs/topi
```

- The following command performs a case insensitive search for the design TEST:

```
rc:/> find . -ignorecase -design test
/designs/TEST
```

- To find hierarchical objects, you can just specify the top-level object instead of the root or current directory. Doing so can provide faster results because it minimizes the number of hierarchies that RTL Compiler traverses. In the following example, if we wanted to only find the output pins for `inst1`, the first specification is more efficient than the second. The second example not only traverses more hierarchies, it also returns `inst2` instances.

```
rc:/> find inst1 -pin out*
{/designs/woodward/instances_hier/inst1/pins_out/out1[3]}
rc:/>find / -pin out*
{/designs/woodward/instances_hier/inst1/pins_out/out1[3]}
{/designs/woodward/instances_hier/inst2/pins_out/out1[3]}
```

- The following example uses the `find` command to return a list of all the instances in a small design.

```
rc:/> find / -instance *
/designs/MOD69/instances_hier/inst1 /designs/MOD69/instances_
hier/inst1/instances_comb/g21
```

The `-vname` option removes the container directories (in this case `instance_hier`) and presents the list more concisely:

```
rc:/> find / -instance -vname *
inst1
inst1/g21
```

This option is useful when you want to present the object in a report because the name is more concise. The disadvantage of the shortened name is that it may no longer refer to a unique object because an instance, pin, net, and support may all share the same Verilog name.

Command Reference for Encounter RTL Compiler

Navigation

- The following example uses the `-regexp` option to return all message objects that contain at least VLOGPT-6:

```
rc:/> find / -regexp (VLOGPT-6+) -messsages * *
```

- The following example returns all combinational instances named g58 or g59 in the Verilog name style:

```
rc:/> find / -regexp {g[5][8-9]} -vname -instance instances_comb/*
```

- Use the `ls -dir` command to format the output row over row:

```
rc:/> ls -dir [find / -instance -vname *]  
/designs/quea/instances_hier/inst1/  
/designs/quea/instances_hier/inst1/instances_comb/g41/  
/designs/quea/instances_hier/inst1/instances_comb/g42/  
/designs/quea/instances_hier/inst1/instances_comb/g43/  
/designs/quea/instances_hier/inst1/instances_comb/g44/
```

- Use the `ls -dir` command and the redirect arrow to redirect the output to the specified file:

```
rc:/> ls -dir [find / -instance -vname *] > quea.txt
```

- You can also append arrows ("`>>`").

- The following examples show the difference between the result with and without `-split`.

```
rc:/designs> set rc [find / -inst a_reg_*]  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_0  
/designs/DTMF_CHIP/instances_hier//DMA_INST/instances_seq/a_reg_1  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_2  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_3  
rc:/designs> echo $rc  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_0  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_1  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_2  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_3  
rc:/designs> set rc [find / -split -inst a_reg_*]  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_3  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_2  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_1  
/designs/DTMF_CHIP/instances_hier/DMA_INST/instances_seq/a_reg_0  
rc:/designs> echo $rc  
rc:/designs>
```

Related Information

Related command: [clock_ports](#) on page 372

Related attributes [find_inefficient_threshold](#)

[find_inefficient_use](#)

inout_mate

```
inout_mate {subport_bus|port_bus|subport|port|pin|pgpin}
```

Distinguishes bidirectional pins, ports, port_busses, supports, and subport_busses inout objects so you can find the input object and the corresponding output object.

Note: The command will return NULL for a pgpin.

These bidirectional inout objects are represented as two distinct objects: the input pin, port, port_bus, subport, and subport_bus object and the corresponding output object. The pair cannot be broken apart, such as using an `edit_netlist rm` command to delete just one of the two objects, because together, they represent the inout object.

Examples

- The following example is a design called `top` that has a primary inout port named `io`, which RC Encounter represents as two distinct ports:

```
/designs/top/ports_in/io  
/designs/top/ports_out/io
```

- Use the `inout_mate` command on the input object to find the corresponding output object:

`rc:/> inout_mate ports_in/io`
`/designs/top/ports_out/io`
 - Use the `inout_mate` command on the output object to find the corresponding input object

`rc:/> inout_mate ports_out/io`
`/designs/top/ports_in/io`

- The following example is an hierarchical instance called `s4` that has an inout port named `io`, which RC Encounter represents as two distinct subports:

```
/designs/top/instances_hier/s4/subports_in/io  
/designs/top/instances_hier/s4/subports_out/io
```

- Use the `inout_mate` command on the input object to find the corresponding output object:

`rc:/> inout_mate s4/subports_in/io`
`/designs/top/instances_hier/s4/subports_out/io`
 - Use the `inout_mate` command on the output object to find the corresponding input object:

`rc:/> inout_mate s4/subports_out/io`
`/designs/top/instances_hier/s4/subports_in/io`

II

`ll virtual_dir`

Lists the contents of the specified virtual directory in long format

A virtual directory is a directory in the design hierarchy.

Options and Arguments

<code>virtual_dir</code>	Specifies the virtual directory whose contents to list. If no directory is specified, the contents of the current directory in the design hierarchy is listed.
--------------------------	---

Examples

- The following example lists the contents of the `counter` (design) directory.

```
rc:/> ll counter
/designs/counter:
Total: 17 items
./                                (design)
constants/
dex_settings/
dft7
instances_comb/
instances_hier/
instances_seq/
modes/
nets/
physical/
port_busses_in/
port_busses_out/
ports_in/
ports_out/
power/
subdesigns/
timing/
rc:/>
```

Related Information

Affected by this command: [cd](#) on page 37

Related command: [pwd](#) on page 56

Command Reference for Encounter RTL Compiler

Navigation

ls

```
ls [-computed] [-attribute] [-long] [-dir] [-R]
    [-width integer] [object]... [>file]
```

Lists information about any objects in the design hierarchy (designs, library cells, clocks, and so on). This command is similar to its UNIX counterpart.

Options and Arguments

-attribute	List the attributes for the specified object whose values are different from the default values.
-computed	Lists all computed attributes. Computed attributes are potentially very time consuming to process and are therefore by default not listed.
-dir	Lists only the directory name not its contents.
<i>file</i>	Specifies the name of the file to which to list the information.
-long	Lists the contents (long listing) of the directory.
<i>object</i>	Specifies the directory for which you want to list information. <i>Default:</i> current directory
-R	Recursively lists the encountered subdirectories.
-width <i>integer</i>	Specifies the width of the screen that can be used to show the information <i>Default:</i> 80

Examples

- The following command lists the contents of the libraries directory:

```
rc:/> ls libraries
/libraries:
./           em333s/           sm333s/
```

- The following command shows information of cell buf1 in regular and long listing format:

```
rc:/libraries/tutorial/libcells/buf1> ls
./             A                 Y
rc:/libraries/tutorial/libcells/buf1> ls -long
Total: 3 items
./                  (libcell)
A                  (libpin)
Y                  (libpin)
```

Command Reference for Encounter RTL Compiler

Navigation

- The following command lists only the attributes of buf1 whose values are different from the default values:

```
rc:/libraries/tutorial/libcells/buf1> ls -attribute
Total: 3 items
./                                (libcell)
Attributes:
  area = 1
  buffer = true
  cell_leakage_power = 0.0 nW
  combinational = true
  liberty_attributes = area 1.0
A/                               (libpin)
Attributes:
  capacitance = 25.0 25.0 femtofarads
  fanout_load = 1.000 fanout_load units
  input = true
  liberty_attributes = capacitance 0.025 direction input
  max_transition = 4500.0
  outgoing_timing_arcs = /libraries/tutorial/libcells/buf1/Y/inarcs/A_n90
Y/                               (libpin)
Attributes:
  capacitance = 0.0 0.0 femtofarads
  fanout_load = 0.000 fanout_load units
  function = A
  incoming_timing_arcs = /libraries/tutorial/libcells/buf1/Y/inarcs/A_n90
  liberty_attributes = direction output function A
  max_transition = 4500.0
  output = true
```

- The following command lists all attributes (except for the computed attributes) for buf1, even those with the default value:

```
rc:/libraries/tutorial/libcells/buf1> ls -long -attribute
Total: 3 items
./                                (libcell)
All attributes:
  adder = false
  area = 1.0
  async_clear =
  async_preset =
  avoid = false
  buffer = true
  cell_delay_multiplier = 1.0
  cell_leakage_power = 0.0 nW
  clock =
  clock_gating_integrated_cell =
  combinational = true
  constraint_multiplier = 1.0
  flop = false
  height = no_value
  ...
  usable = true
  user_defined =
  width = no_value
A                               (libpin)
All attributes:
  async_clear_phase = none
  async_preset_phase = none
  capacitance = 25.0 25.0 femtofarads
```

Command Reference for Encounter RTL Compiler

Navigation

```
clock_gate_clock_pin = false
clock_gate_enable_pin = false
clock_gate_obs_pin = false
clock_gate_out_pin = false
clock_gate_reset_pin = false
clock_gate_test_pin = false
...
tristate = false
user_defined =
Y          (libpin)
All attributes:
async_clear_phase = none
async_preset_phase = none
capacitance = 0.0 0.0 femtofarads
clock_gate_clock_pin = false
clock_gate_enable_pin = false
clock_gate_obs_pin = false
clock_gate_out_pin = false
clock_gate_reset_pin = false
clock_gate_test_pin = false
...
tristate = false
user_defined =
```

- The following command uses wildcard strings and the results of a find command:

```
rc:/libraries> ls -long [find . -libcell A*]
/libraries/cg/libcells/AND2A:
Total: 4 items
./          (libcell)
A/          (libpin)
B/          (libpin)
Z/          (libpin)

/libraries/cg/libcells/AO21A:
Total: 5 items
./          (libcell)
A/          (libpin)
B/          (libpin)
C/          (libpin)
Z/          (libpin)
```

- The following command shows that the computed attribute `timing_case_computed_value` has been turned on:

```
rc:/>ls -computed /designs/violet/instances_comb/U1/pins_in/S
timing_case_computed_value = 1
```

- The following examples show the difference between the `ls -attribute` and `get_attribute` commands.

```
rc:/designs> ls -attribute
Total: 2 items
./
async_set_reset_flop_n/          (design)
Attributes:
    dft_mix_clock_edges_in_scan_chains = false
    wireload = /libraries/slow/wireload_models/sartre18_Conervative

rc:/designs> get_attribute wireload /designs/async_set_reset_flop_n/
/libraries/slow/wireload_models/sartre18_Conervative
```

Command Reference for Encounter RTL Compiler

Navigation

The `ls -attribute` command lists all *user-modified* attributes and their values. The `get_attribute` command lists only the value of the specified attribute. The `get_attribute` command is especially useful in scripts where its returned values can be used as arguments to other commands.

- The following command lists all subdirectories in directory `sdp_groups`.

```
rc:/designs/test/sdp_groups> ls -R
./
ga/
ga/sdp_columns/
ga/sdp_datapaths/
ga/sdp_rows/
ga/sdp_rows/rb/
ga/sdp_rows/rb/sdp_columns/
ga/sdp_rows/rb/sdp_instances/
ga/sdp_rows/rb/sdp_instances/skip_instance_0
ga/sdp_rows/rb/sdp_instances/skip_instance_1
ga/sdp_rows/rb/sdp_instances/st_box1_g1
ga/sdp_rows/rb/sdp_instances/st_box2_g1
ga/sdp_rows/skip_row_0/
ga/sdp_rows/skip_row_0/sdp_columns/
ga/sdp_rows/skip_row_0/sdp_instances/
rc:/designs/test/sdp_groups>
```

Related Information

Related command: [get_attribute](#) on page 77

popd

`popd`

Removes the topmost element of the directory stack, revealing a new top element and changes the current directory to the new top element. This command is similar to its UNIX counterpart.

Note: If `popd` is issued on a directory stack that has only one element, an appropriate warning message is printed and the `popd` command exits without changing the directory stack.

Examples

- In the following example, the directory stack starts off as `/libraries /designs`: `/libraries` is the current directory and the top element of the directory stack, and `/designs` is next on the stack). When the `popd` command is issued, `/libraries` is popped off, and `/designs` becomes the top (and only element) of the stack and the current directory.

```
rc:/libraries> dirs  
/libraries /designs  
rc:/libraries> popd  
/designs  
rc:/designs> dirs  
/designs  
rc:/designs> pwd  
/designs
```

- In the following example, a `popd` command is issued on a directory stack that has only one element.

```
rc:/> cd /designs  
rc:/designs> dirs  
/designs  
rc:/designs> popd  
Directory stack empty  
/designs
```

This can happen when more `popd` commands are issued than `pushd` commands.

Related Information

Affects these commands:

[dirs](#) on page 40

[ls](#) on page 50

[pushd](#) on page 55

[pwd](#) on page 56

pushd

`pushd directory`

Pushes the specified new target directory onto the directory stack (as the topmost element) and changes the current directory to that specified directory. This command is similar to its UNIX counterpart.

Options and Arguments

directory Specifies the name of the directory to be set as target directory.
You can use wildcards when they do not produce an ambiguous reference (more than one match).

Examples

- In the following example, /libraries is pushed onto the top of the /designs directory stack and the current directory is changed to it:

```
rc:/designs> pushd /libraries  
/libraries /designs  
rc:/libraries>
```

- In the following example, the push operation does not succeed because there is more than one directory that starts with ex.

```
rc:/libraries> pushd ex*  
Error : A single object was expected, but multiple objects were found. [TUI-62]  
: The argument that found multiple objects was 'ex*'.  
pushd: push current dir onto stack and cd to new dir  
Usage: pushd <object>  
<object>:  
      new target directory  
Failed on pushd ex*
```

Related Information

Affects these commands:

[dirs](#) on page 40

[ls](#) on page 50

[popd](#) on page 54

[pwd](#) on page 56

pwd

pwd

Displays the current position in the design hierarchy. This command is similar to its UNIX counterpart.

Examples

- The following example shows the current directory after changing the current directory first:

```
rc:/> cd /designs  
rc:/designs> pwd  
/designs
```

Related Information

Related commands:	<u>dirs</u> on page 40
	<u>ls</u> on page 50
	<u>popd</u> on page 54
	<u>pushd</u> on page 55

vdir_lsearch

```
vdir_lsearch list object
```

Performs an lsearch of a *vdir* type object in a list of objects. If a match is found, it returns the position of that object in the list. If no match is found, it returns -1.

While performing the search, the command performs a (fast) direct comparison of every *vdir* type object in the list with the pointer of the given object; whereas for every non-*vdir* type object in the list, it creates a string-name of the object and compares it with the string-name of the given object. This makes it faster than the regular lsearch command, which does string-name derivation and string comparison for every object in the list.

Note: Examples of commands that return *vdir* type objects are `find`, `edit_netlist` and `get_attribute`.

Options and Arguments

<i>list</i>	Specifies a list of objects. The objects can be regular objects or can be of type <i>vdir</i> .
<i>object</i>	Specifies the object for which you want to know the position in the list. The object must be of type <i>vdir</i> . Note: If the object is not of type <i>vdir</i> , a regular lsearch will be performed.

Example

In the following example, the first search, performed on a list of *vdir* type objects (result of `find` command), returns -1. To this list, two regular objects are added: `abc` and `/designs/test/instances_hier/mux_oo_5_10/pins_out/z`. The search on this mixed list of objects returns 4.

```
rc:/> set pin [get_attr driver /designs/test/nets/oo]
/designs/test/instances_hier/mux_oo_5_10/pins_out/z
rc:/> set list [find / -pin in*]
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0
rc:/> vdir_lsearch $list $pin
-1
rc:/> lappend list "abc"
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0 abc
rc:/> lappend list "/designs/test/instances_hier/mux_oo_5_10/pins_out/z"
```

Command Reference for Encounter RTL Compiler

Navigation

```
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_0
/designs/test/instances_hier/mux_oo_5_10/pins_in/in_1
/designs/test/instances_comb/g4/pins_in/in_0 abc
/designs/test/instances_hier/mux_oo_5_10/pins_out/z
rc:/> vdir_lsearch $list $pin
4
```

Command Reference for Encounter RTL Compiler

Navigation

vname

```
vname [instance | net | pin | pgpin | port | subport  
| subdesign | design]...
```

Returns the Verilog name of the specified objects.

Note: You can specify a list of objects and the tool returns a list.

Example

The following command returns the Verilog name of instance a_reg_3 .

```
vname [find / -inst a_reg_3]  
DTMF_INST/DMA_INST/a_reg_3
```

what_is

what is object

Returns a string describing the type of object given as its argument. The command will work only if a single object is specified. A list of valid objects can be obtain by typing `find -help`.

This command is mostly used for writing Tcl scripts (rather than being an interactive command). It is useful for checking the types of arguments to the Tcl procedures.

Options and Arguments

object Specifies the object for which you want to know the type.

- The following command fails because there is more than one object of the given name in the current tree structure.

```
rc:/designs/alu> what_is alu*
Error : A single object was expected, but multiple objects were found. [TUI-62]
          : The argument that found multiple objects was 'alu*'.
  what_is: return an object's type
Usage: what_is <object>
      <object>:
          drs object of interest
Failed on what is alu*
```

what_is_list

```
what_is_list  
object...
```

Returns a list of strings describing the object types of the specified object(s).

Options and Arguments

<i>object</i>	Specifies the object whose types you want to know.
---------------	--

Examples

- The following command returns the object types for CLK1.

```
rc:/> what_is_list CLK1  
exception cost_group
```

Using the `find` command returns the paths to these object types:

```
rc:/> find / * CLK1  
/designs/top/timing/clock_domains/domain_1/CLK1  
/designs/top/timing/exceptions/path_groups/CLK1 /designs/top/timing/cost_  
groups/CLK1
```

Using the `what_is` command would fail because multiple object types are found for CLK1:

```
rc:/> what_is CLK1  
Error : A single object was expected, but multiple objects were found. [TUI-  
62] [what_is]  
      : The argument that found multiple objects was 'CLK1'.  
what_is: returns an object's type  
  
Usage: what_is <object>  
  
<object>:  
      object of interest  
Failed on what_is CLK1
```

Related Information

Related command: [what_is](#) on page 60

Command Reference for Encounter RTL Compiler

Navigation

General

- [? on page 65](#)
- [alias on page 66](#)
- [all_inputs on page 67](#)
- [all_outputs on page 68](#)
- [apropos on page 69](#)
- [attribute_exists on page 70](#)
- [clear on page 71](#)
- [date on page 72](#)
- [enable_transparent_latches on page 73](#)
- [exec_embedded_script on page 74](#)
- [exit on page 76](#)
- [get_attribute on page 77](#)
- [get_liberty_attribute on page 79](#)
- [get_read_files on page 80](#)
- [help on page 81](#)
- [include on page 82](#)
- [lcd on page 83](#)
- [license on page 84](#)
- [license_checkin on page 85](#)
- [license_checkout on page 86](#)
- [license_feature on page 87](#)

Command Reference for Encounter RTL Compiler

General

- [license list](#) on page 88
- [license version](#) on page 89
- [lls](#) on page 90
- [lpopd](#) on page 91
- [lpushd](#) on page 92
- [lpwd](#) on page 93
- [man](#) on page 94
- [more](#) on page 95
- [quit](#) on page 97
- [rc](#) on page 98
- [redirect](#) on page 103
- [reset_attribute](#) on page 105
- [resume](#) on page 106
- [sdc_shell](#) on page 107
- [set_attribute](#) on page 108
- [shell](#) on page 110
- [stop_suspend](#) on page 111
- [string_representation](#) on page 112
- [suppress_messages](#) on page 113
- [suspend](#) on page 114
- [tcl_load](#) on page 115
- [unsuppress_messages](#) on page 118

Command Reference for Encounter RTL Compiler

General

?

? [command]...[> file]

Provides help on the specified RTL Compiler commands.

Note: You can get help at any stage of the design.

Options and Arguments

command

Specifies the command for which you want help.

If you do not specify a command name, you get a brief summary of all RTL Compiler commands.

file

Specifies the name of the file to which to write the help.

Examples

- The following example requests help for the

```
rc:/> ? all_inputs  
That command is:
```

```
    General  
=====  
all_inputs  returns all the input ports.
```

Command details:

```
    all_inputs: returns all the input ports.
```

```
Usage: all_inputs [-design <design>]
```

```
    [-design <design>]:  
        limits list of input ports to the specified top-level design
```

- The following example requests help for the `synthesize` and `report` commands:

```
rc:/> ? synthesize report  
Commands are:
```

```
    Analysis  
=====  
report      generates one of various reports
```

```
    Synthesis  
=====  
synthesize  synthesizes the design
```

alias

```
alias alias_name command_name
```

Defines an alias for the specified name.

Options and Arguments

<i>alias_name</i>	Specifies the alias name.
<i>command_name</i>	Specifies the name of the command for which you want to create an alias.

Example

The following command creates an alias ai for the all_inputs command.

```
rc:/> alias ai all_inputs
ai
rc:/> ai -h
    all_inputs: returns all the input ports.

Usage: all_inputs [-design <design>]
        [-design <design>]:
            limits list of input ports to the specified top-level design
```

all_inputs

```
all_inputs [-design design]
```

Returns the input ports of the specified design.

Options and Arguments

-design design Specifies the name of the top-level design for which you want to list all input ports.

If you omit the design name, the input ports of all loaded designs are listed.

Examples

- The following example lists the input ports of all loaded designs.

```
rc:/designs/top> all_inputs
/designs/top/ports_in/enable{/designs/top/ports_in/in1[7]}
{/designs/top/ports_in/in1[6]}{/designs/top/ports_in/in1[5]}
{/designs/top/ports_in/in1[4]}{/designs/top/ports_in/in1[3]}
{/designs/top/ports_in/in1[2]}{/designs/top/ports_in/in1[1]}
{/designs/top/ports_in/in1[0]}{/designs/top/ports_in/in2[7]}
{/designs/top/ports_in/in2[6]}{/designs/top/ports_in/in2[5]}
{/designs/top/ports_in/in2[4]}{/designs/top/ports_in/in2[3]}
{/designs/top/ports_in/in2[2]}{/designs/top/ports_in/in2[1]}
{/designs/top/ports_in/in2[0]}/designs/top/ports_in/clk
/designs/my_CG_MOD/ports_in/ck_in /designs/my_CG_MOD/ports_in/enable
/designs/my_CG_MOD/ports_in/test /designs/my_CG_MOD_neg/ports_in/ck_in
/designs/my_CG_MOD_neg/ports_in/enable /designs/my_CG_MOD_neg/ports_in/test
```

- The following example lists the input ports of design my_CG_MOD.

```
rc:/designs/top> all_inputs -design my_CG_MOD
/designs/my_CG_MOD/ports_in/ck_in /designs/my_CG_MOD/ports_in/enable /
designs/my_CG_MOD/ports_in/test
```

- Use the **ls -dir** command to format the output of the command

```
rc:/> ls -dir [all_inputs]
/designs/ksable/ports_in/in1[0]
/designs/ksable/ports_in/in1[1]
/designs/ksable/ports_in/in1[2]
```

- Use the **ls -dir** command with the redirect arrow to redirect the output to a specified file:

```
rc:/> ls -dir [all_inputs] > areid.txt
```

You can also append arrows (">>").

all_outputs

```
all_outputs [-design design]
```

Returns the output ports of the specified design.

Options and Arguments

-design *design* Specifies the name of the top-level design for which you want to list all output ports.

If you omit the design name, the output ports of all loaded designs are listed.

Examples

- The following example lists the output ports of all loaded designs.

```
rc:/designs/top> all_outputs
{/designs/top/ports_out/out1[7]}{/designs/top/ports_out/out1[6]}
{/designs/top/ports_out/out1[5]}{/designs/top/ports_out/out1[4]}
{/designs/top/ports_out/out1[3]}{/designs/top/ports_out/out1[2]}
{/designs/top/ports_out/out1[1]}{/designs/top/ports_out/out1[0]}
{/designs/top/ports_out/out2[7]}{/designs/top/ports_out/out2[6]}
{/designs/top/ports_out/out2[5]}{/designs/top/ports_out/out2[4]}
{/designs/top/ports_out/out2[3]}{/designs/top/ports_out/out2[2]}
{/designs/top/ports_out/out2[1]}{/designs/top/ports_out/out2[0]}
/designs/my_CG_MOD/ports_out/ck_out /designs/my_CG_MOD_neg/ports_out/ck_out
```

- The following example lists the output ports of design my_CG_MOD.

```
rc:/designs/top> all_outputs -design my_CG_MOD
/designs/my_CG_MOD/ports_out/ck_out
```

- Use the **ls -dir** command to format the output of the command

```
rc:/> ls -dir [all_inputs]
/designs/ksable/ports_out/out1[0]
/designs/ksable/ports_out/out1[1]
/designs/ksable/ports_out/out1[2]
```

- Use the **ls -dir** command with the redirect arrow to redirect the output to a specified file:

```
rc:/> ls -dir [all_outputs] > areid.txt
```

You can also append arrows (">>").

apropos

```
apropos [-skip_help] [-skip_commands]  
[-skip_attributes] string
```

Performs a case insensitive wildcard search of commands (including their options) and attributes. The command will even encompass the help text of commands and attributes. The search results will be categorized into the following different sections:

- Commands that match search text
- Commands with help text that match search text
- Commands with options (both option name and option help text) that match search text
- Attributes that match search text
- Attributes with help text and attribute objects that matches search text

Options and Arguments

<code>-skip_help</code>	Do not include help text in the search results.
<code>-skip_command</code>	Do not include commands in the search results.
<code>-skip_attributes</code>	Do not include attributes in the search results.
<code>string</code>	Specifies the search string

Examples

- The following example searches for the term generic:

```
rc:/> apropos generic  
Commands with option text matching search string:  
  synthesize  
  write_hdl  
  
Attributes with help text matching search string:  
  dp_perform_csa_operations (root)  
  dp_perform_sharing_operations (root)  
  dp_perform_speculation_operations (root)  
  hdl_auto_sync_set_reset (root)  
  timing_driven_muxopto (design)  
  timing_driven_muxopto (subdesign)
```

- The following example searches for the term generic among commands only:

```
rc:/> apropos -skip_attributes generic  
Commands with option text matching search string:  
  synthesize  
  write_hdl
```

attribute_exists

```
attribute_exists attribute
  {-path object_paths | -type object_type}
```

Checks if the specified attribute exists.

A 1 is returned when the attribute exists, a 0 when the attribute does not exist.

Options and Arguments

<i>attribute</i>	Specifies the attribute name to be checked.
<i>-path object_path</i>	Specifies the path to the object for which the attribute must be checked.
<i>-type object_type</i>	Specifies the object type for which the attribute must be checked.

Examples

- The following command checks whether `osc_source` exists for instances.

```
rc:/> attribute_exists osc_source -type instance
0
```

- The following command checks whether `count` exists for message TUI-20.

```
rc:/> attribute_exists count -path /messages/TUI/TUI-20
1
```

Command Reference for Encounter RTL Compiler

General

clear

clear

Clears the terminal screen.

date

date

Returns the date and time. This command is equivalent to the UNIX date command.

Examples

```
rc:/> date  
Thu Apr 23 01:28:55 PM PDT 2009
```

enable_transparent_latches

`enable_transparent_latches`

Enables transparent latches in the design by disabling the EN to Q arcs in the latch. This command must be used after `elaborate`. Transparent latches are latches with the enable signal held constant at the active state. Without enabling transparent latches, paths through them cannot be traced.

Related Information

Affects this command: [report_disabled_transparent_latches](#) on page 426

exec_embedded_script

```
exec_embedded_script  
  [-design string] [-subdesign string]
```

Executes embedded scripts found in a specified design or subdesign. To execute the scripts on all top-designs and their subdesigns, use the `exec_embedded_script` command without any arguments. Use this command after the `elaborate` command.

The embedded script of a design or subdesign is stored in the `embedded_script` attribute of that design or subdesign.

Options and Arguments

<code>-design <i>string</i></code>	Specifies the top level design for which the embedded script need to be executed. Using this option executes scripts for the specified design and every subdesign in it.
<code>-subdesign <i>string</i></code>	Specifies the full path of the subdesign for which the embedded script needs to be executed. Using this option executes the script only for the specified subdesign.

Examples

- The following commands execute a SDC script, if any, embedded in the RTL description of the design.

```
rc> exec_embedded_script  
rc> exec_embedded_script -design name of top design  
rc> exec_embedded_script -design full vdir path to top design  
rc> exec_embedded_script -subdesign full vdir path to subdesign
```
- If design or subdesign is not specified, then using this command executes the embedded SDC script of all designs and all subdesigns.
- If a design is specified, then this command executes the embedded SDC script in the RTL code of the given design and all its subdesigns.
- If a subdesign is specified, then this command executes the embedded SDC script in the RTL code of the specific subdesign only.

Note: The impact of executing only this embedded script can still be hierarchical. For example, this can happen if an SDC command in the embedded script at this level of design hierarchy specifies a constraint for some instance down in the hierarchy.

- If both a design and a subdesign is specified, then the subdesign setting is ignored.

Command Reference for Encounter RTL Compiler

General

Related Information

Related command: [elaborate](#) on page 332

Related attributes: [hdl_auto_exec_sdc_scripts](#)

exit

`exit [code]`

Exits from the RTL Compiler shell without saving design data.

Control-c is a shortcut to the `exit` and `quit` commands.



When exiting, no design data is saved, so it is important to save the design using the `write_hdl` and `write_script` commands.

Options and Arguments

`code`

Specifies the exit code.

The following are the built-in exit codes:

0 – normal exit

1 – abnormal exit

246 – exit when no license server is available

245 – exit when no license feature is available

244 – exit when syntax error in script

get_attribute

```
get_attribute {attribute_name [object]
   | -h {attribute_name|*} {type|*} }
```

Retrieves the value of an attribute set by RTL Compiler or via the `set_attribute` command.

You can also use this command to list

- All attributes associated with a given object type
- All objects to which the specified attribute applies

This command is most commonly used within scripts to control the way a script operates based on the value of the attribute, or to retrieve basic information on the tool.

Options and Arguments

<i>attribute_name</i>	Specifies the name of an attribute whose value you want to retrieve.
<i>object</i>	Specifies the path to the object for which the attribute value should be retrieved. <i>Default:</i> current working directory
<i>type</i>	Specifies the object type for which you want the list of attribute names. Check the command help for a list of the valid object types. Note: Some of the object types currently do not have any attributes associated with them.

Examples

- The following example retrieves the current value of the `library` attribute on the root directory:

```
rc:/> get_attribute library /
tutorial.lbr
```

- The following example assumes you are already at the root of the design hierarchy, so the object specification is omitted:

```
rc:/> get_attribute library
tutorial.lbr
```

Command Reference for Encounter RTL Compiler

General

- The following example stores the attribute value in a variable:

```
rc:/> set old_library [get_attribute library /]  
tutorial.lbr
```

- The following example returns the area of library cell:

```
rc:/> get_attribute area [find /lib* -libcell nor*]  
1.5
```

- The following example lists all root-level attributes starting with lp:

```
rc:/> get_attribute lp* root -help
```

- The following example lists all attributes for all object types:

```
rc:/> get_attribute * * -help
```

- The following examples show the difference between the **ls -attribute** and **get_attribute** commands.

- Using the **ls -attribute** command:

```
rc:/designs> ls -attribute  
Total: 2 items  
. /  
async_set_reset_flop_n/ (design)  
Attributes:  
    dft_mix_clock_edges_in_scan_chains = false  
    wireload = /libraries/slow/wireload_models/sartre18_Conervative
```

- Using the **get_attribute wireload** command:

```
rc:/designs> get_attribute wireload /designs/async_set_reset_flop_n/
```

returns the following wireload model:

```
/libraries/slow/wireload_models/sartre18_Conervative
```

The **ls -attribute** command lists all user modified attributes and their values. The **get_attribute** command lists only the value of the specified attribute. The **get_attribute** command is especially useful in scripts where its returned values can be used as arguments to other commands.

Related Information

Affected by this command: [set_attribute](#) on page 108

Related command: [ls](#) on page 50

get_liberty_attribute

```
get_liberty_attribute attribute_name
    {libarc|libcell|libpin|library|operating_condition|wireload}
```

Options and Arguments

attribute_name Specifies the name of the Liberty attribute whose value you want to retrieve.

libarc|libcell|libpin|library|operating_condition|wireload

Specifies the name of the object for which you want to retrieve a Liberty attribute value.

Example

The following command retrieves the default_operating_conditions attribute of the tutorial library.

```
rc:/> get_liberty_attribute default_operating_conditions tutorial
typical_case
```

get_read_files

```
get_read_files [-quiet] [-command command]
```

Returns information on the files that have been read for the specified command

Options and Arguments

- command *command* Specifies the command for which you want the information.
- quiet Suppresses the status message.

Example

```
rc:/> get_read_files -command read_def  
point.def
```

help

```
help [command]... [> file]
```

Provides help on the specified RTL Compiler commands.

Note: You can get help at any stage of the design.

Options and Arguments

command Specifies the command for which you want help.

If you do not specify a command name, you get a brief summary of all RTL Compiler commands.

file Specifies the name of the file to which to write the help.

Examples

- The following example requests help for the `report` command:

```
rc:/> help report
That command is:
    report      generate one of various reports
```

- The following example requests help for the `synthesize` and `report` commands:

```
rc:/> help synthesize report
Commands are:
    report      generate one of various reports
    synthesize  synthesize the design
```

include

```
include file
```

Executes scripts containing RTL Compiler or Tcl commands in the order they are listed in the `include` command.

When RTL Compiler begins executing, a number of scripts are automatically included. Information on these configuration scripts is given in the [Using Encounter RTL Compiler](#).

The use of script files allows automation and modularization of the design flow by placing commonly used commands and sequences of commands into their own scripts. For externally defined components like memories, the vendor can supply a script to create and set up the memory.

This command is identical to the `source` command.

Note: If an error occurs during execution of one of the scripts, execution of that script (and all scripts following it) is stopped.

Options and Arguments

<code>file</code>	Specifies the name of the script file to include.
-------------------	---

Examples

- The following example includes one script file:

```
rc:/> include constraint.g
```

- The following example includes multiple scripts at once:

```
rc:/> include constraint.g; include synth.g
```

Related Information

Affected by these attributes: [hdl_search_path](#)
[lib_search_path](#)
[script_search_path](#)

lcd

lcd *directory*

Changes the UNIX working directory to the specified directory.

Options and Arguments

directory Specifies the UNIX directory to which to change the current directory.

Examples

- The following example changes the working directory to the `rtl_lab01` directory in the current UNIX directory.

```
rc:>/ lcd rtl lab01
```

Related Information

Affects these commands:

lls on page 90

Ipwd on page 93

[shell](#) on page 110

license

```
license {checkin | checkout | feature | list | version}
```

Manages the checking in and checking out of licenses.

Options and Arguments

checkin	Checks in a product license that was previously checked out.
checkout	Checks-out additional licenses.
feature	Checks if the license is available.
list	Returns a list of licenses currently checked out.
version	Returns the checked out license version.

Related Information

Related commands:	license checkin on page 85
	license checkout on page 86
	license feature on page 87
	license list on page 88
	license version on page 89

license checkin

```
license checkin license
```

Checks in a product license that was previously checked out.

Options and Arguments

license Specifies the license to be checked in. The licenses that can be checked in are:

- `RTL_Compiler_CPU_Accel_Option`
- `RTL_Compiler_Ultra`
- `RTL_Compiler_Ultra_II_Option`

Examples

- The following example checks in the `RTL_Compiler_Ultra` license:

```
rc:/> license checkin RTL_Compiler_Ultra
```

license checkout

```
license checkout license [-version float] [-wait]
```

Checks-out an additional product license. Licenses can only be checked-out one at a time.

If the license is available, it is checked out and 1 is returned. If the license is not available, a warning message is issued and 0 is returned. To check-in a license, use the `license checkin` command or quit RTL Compiler.

Options and Arguments

<code>license</code>	Specifies the license to be checked out. The licenses that can be checked out are: <ul style="list-style-type: none">■ <code>RTL_Compiler_Adv_Phys_Option</code>■ <code>RTL_Compiler_CPU_Accel_Option</code>■ <code>RTL_Compiler_Low_Power_Option</code>■ <code>RTL_Compiler_Ultra</code>■ <code>RTL_Compiler_Ultra_II_Option</code>
<code>-version float</code>	Allows you to specify a specific version of the license. By default, the command uses the hardcoded primary or alternate version of the license that you want to check out.
<code>-wait</code>	Waits until a license becomes available. RTL Compiler will wait indefinitely, until the license is available.

Examples

- The following example checks-out the RTL Compiler Ultra license:

```
rc:/> license checkout RTL_Compiler_Ultra  
Checking out license 'RTL_Compiler_Ultra'..... (1 second elapsed)  
1
```

- The following command checks out version 8.2 of the RTL Compiler Ultra II Option license.

license checkout RTL Compiler Ultra II Option -version 8.2

license feature

`license feature license [-version float]`

Checks if the specified license feature product license feature exists. The command returns “1” if the specified license feature is available, and “0” if it is not available.

Use this command in scripts to check a license before checking it out.

Options and Arguments

<i>license</i>	Specifies the license to be checked. The licenses that can be checked in are: <ul style="list-style-type: none">■ RTL_Compiler_Adv_Phys_Option■ RTL_Compiler_CPU_Accel_Option■ RTL_Compiler_Low_Power_Option■ RTL_Compiler_Ultra■ RTL_Compiler_Ultra_II_Option
<i>-version float</i>	Specifies the license version.

Examples

- The following example checks if the RTL Compiler Ultra product license exist:

```
rc:/> license feature RTL_Compiler_Ultra
```

The returned value "1" indicates that the license is available.

license list

`license list`

Returns a Tcl list of additional licenses that are currently checked-out. The license used to launch RTL Compiler is not included in this list.

Example

- The following example shows that one additional license is currently checked-out.

```
rc:/> license list  
RTL_Compiler_Low_Power_Option
```

license version

`license version license`

Returns the actual version of the license checked out.

Options and Arguments

license

Specifies the license for which to check the version. You can specify any of the following:

- `RTL_Compiler_Adv_Phys_Option`
- `RTL_Compiler_CPU_Accel_Option`
- `RTL_Compiler_Low_Power_Option`
- `RTL_Compiler_Ultra`
- `RTL_Compiler_Ultra_II_Option`

11s

`lls directory`

Lists the contents of the specified UNIX directory.

Options and Arguments

directory Specifies the UNIX directory whose contents to list.

Examples

- The following example lists the contents of the `rtl_lab01` directory in the current UNIX directory.

```
rc:/> lls rtl_lab01  
alu.v  
lab01_gates.v  
rc.cmd  
rc.log  
script.g  
tutorial.lbr  
rc:/>
```

Related Information

Affected by this command: [lcd](#) on page 83

Related commands: [lpwd](#) on page 93

[shell](#) on page 110

lpopd

lpopd

Removes the topmost element of the UNIX directory stack, revealing a new top element and changes the current directory to the new top element. This command is equivalent to the UNIX popd command.

Note: If lpopd is issued on a directory stack that has only one element, an appropriate warning message is printed and the lpopd command exits without changing the directory stack.

Example

In the following example, the synthesis directory is removed from the UNIX directory stack.

```
rc:/> lpwd  
/home/ria/rc_ex/synthesis  
rc:/> lpopd  
/home/ria/rc_ex
```

Related Information

Affects these commands:	lls on page 90 lpushd on page 92 lpwd on page 93
-------------------------	--

Ipushd

`lpushd directory`

Pushes the specified new target directory onto the UNIX directory stack (as the topmost element) and changes the current directory to that specified directory. This command is equivalent to the UNIX `pushd` command.

Options and Arguments

directory Specifies the name of the UNIX directory to be set as target directory.

You can use wildcards when they do not produce an ambiguous reference (more than one match).

Example

In the following example, the synthesis directory is pushed on top of the current UNIX directory.

```
rc:/> lpwd  
/home/ria/rc_ex  
rc:/> lpushd synthesis  
synthesis  
rc:/> lpwd  
/home/ria/rc_ex/synthesis
```

Related Information

Affects these commands:

[lls on page 90](#)
[lpopd on page 91](#)
[lpwd on page 93](#)

lpwd

`lpwd`

Returns the UNIX working directory.

Examples

- The following example shows the current UNIX directory.

```
rc:/> lpwd  
/usr3/verilog_labs
```

Related Information

Affected by this command: [lcd on page 83](#)

Related commands: [lls on page 90](#)
[shell on page 110](#)

Command Reference for Encounter RTL Compiler

General

man

```
man { command_name | attribute_name | message_ID}
```

Returns detailed information about the specified command, attribute, or message from the online reference manual. You must first set your `MANPATH` environment variable to the following path:

```
$CDN_SYNTH_ROOT/share/synth/man
```

After setting the `MANPATH` variable, you can obtain manpages for commands and attributes both within RTL Compiler or within the Unix shell. To obtain more information about RTL Compiler messages, you must use the `man` command within RTL Compiler.

Example

- The following example returns the manpage for the `synthesize` command:

```
rc:/> man synthesize
User Commands                               synthesize(1)
```

NAME

```
      synthesize
```

SYNTAX

```
      synthesize [-effort {high|medium|low}] [-to_generic]
                  [-to_mapped] [-incremental] [<design>]...
```

DESCRIPTION

Determines the most suitable design implementation using the given design constraints (clock cycle, input delays, output delays, technology library, and so on).

```
...
```

- The following example returns information about the `TIM-11` message:

```
rc:/> man TIM-11
Entry       : TIM-11
Severity    : Warning
Verbosity   : Message is visible at any 'information_level' above '1'.
Description : Possible timing problems have been detected in this design.
Help        : Use 'report timing -lint' for more information
```

more

`more command`

Displays the output of the specified command one screen at a time. This command is similar to the UNIX more command.

If the output is several screens, the percentage of characters displayed so far is also shown at the bottom of the screen.

To scroll down the display one line at a time, press *Return*.

To display the next screen, press the SPACE bar.

To stop the display, press *q*.

Options and Arguments

`command` Specifies the command whose output you want to display with the `more` command.

Examples

The following command displays one screen for the output of the `write_scandef` command.

```
rc:/> more write_scandef
VERSION 5.4 ;
NAMECASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN ria_test ;

SCANCHAINS 6 ;
- AutoChain_1_seg1_test_clk1_falling
#    + PARTITION p_test_clk1_falling
#        MAXBITS 5
+ START PIN DFT_sdi_1
+ FLOATING
    out1_reg[4] ( IN SI ) ( OUT QN )
    out1_reg[5] ( IN SI ) ( OUT QN )
    out1_reg[6] ( IN SI ) ( OUT QN )
    out1_reg[7] ( IN SI ) ( OUT QN )
    out1_reg[8] ( IN SI ) ( OUT QN )
+ STOP DFT_lockup_g1 D
;
- AutoChain_1_seg2_test_clk2_falling
```

Command Reference for Encounter RTL Compiler

General

```
#      + PARTITION p_test_clk2_falling
#      MAXBITS 5
+ START DFT_lockup_g1 Q
+ FLOATING
  out2_reg[4] ( IN SI ) ( OUT QN )
  out2_reg[5] ( IN SI ) ( OUT QN )
  out2_reg[6] ( IN SI ) ( OUT QN )
  out2_reg[7] ( IN SI ) ( OUT QN )
  out2_reg[8] ( IN SI ) ( OUT QN )
+ STOP DFT_lockup_g259 D
;

- AutoChain_1_seg3_test_clk3_falling
#      + PARTITION p_test_clk3_falling
#      MAXBITS 5
+ START DFT_lockup_g259 Q
+ FLOATING
  out3_reg[4] ( IN SI ) ( OUT QN )
  out3_reg[5] ( IN SI ) ( OUT QN )
  out3_reg[6] ( IN SI ) ( OUT QN )
  out3_reg[7] ( IN SI ) ( OUT QN )
  out3_reg[8] ( IN SI ) ( OUT QN )
--More-- (51%)
```

- The following command outputs the library check report results:

```
more report checks -library
```

- The following command outputs the HDL lint check report results, displaying every message for every message ID, one message per line:

```
more report checks -hdl_lint -detail
```

quit

`quit`

Exits from the RTL Compiler shell without saving design data.
Control-C is a shortcut to the `exit` and `quit` commands.



When exiting, no design data is saved, so it is important to save the design using the `write_hdl` and `write_script` commands.

Options and Arguments

`code` Specifies the exit code.

The following are the built-in exit codes:

0 – normal exit

1 – abnormal exit

246 – exit when no license server is available

245 – exit when no license feature is available

244 – exit when syntax error in script

Command Reference for Encounter RTL Compiler

General

rc

```
rc [-32 | -64 | -32only | -64only | -3264 | -6432]
    [-quiet3264] [-debug3264] [-plat platform] [-v3264]
    [-help3264] [-bg] [-E]
    [-execute command] [-files file] [-post command]
    [-display string] [-gui | -nogui] [-no_custom]
    [-cmdfile file] [-logfile log_file] [-overwrite]
    [-lsf_cpus integer] [-lsf_queue string]
    [-N integer] [-version]
    [-queue] [-wait integer] [-ctos] [-rcl] [-rcp] [-vdi]
    [-use_license { C_to_Silicon_Compiler_L
        | RTL_Compiler_L | RTL_Compiler_Physical
        | RTL_Compiler_Ultra | RTL_Compiler_Verification
        | Virtuoso_Digital_Implem}] ...
```

Starts RTL Compiler from the UNIX environment. If you specify multiple licenses with the `-use_license`, the tool will use the first one specified that is available.

Note: Use `rc64` instead of `rc` if you are using RTL Compiler on a Solaris 64-bit environment.



Tip

You can abbreviate the options for the `rc` command as long as there are no ambiguities with other options. In the following example, `-ve` implies the `-version`:

```
unix> rc -ve
```

Just using `rc -v` would not work because there is more than one option that starts with the letter "v."

Options and Arguments

-32	Launches RTL Compiler in 32-bit mode. This is the default mode. If the 32-bit version is not available, prints a warning and tries to run the 64-bit version.
-3264	Tries to run the 32-bit version of the application. If the 32-bit version is not available, prints an info and tries to run the 64-bit version.
-32only	Tries to run the 32-bit version of the application. If the 32-bit version is not available, prints an error and exits with an error.

Command Reference for Encounter RTL Compiler

General

-64	Launches RTL Compiler in 64-bit mode. Note: You can set the CDS_AUTO_64BIT environment variable to ALL to launch not only RTL Compiler, but all Cadence tools in 64-bit. You will not need to specify the -64 option if you use this variable.
-6432	Tries to run the 64-bit version of the application. If the 64-bit version is not available, prints an info and tries to run the 32-bit version.
-64only	Tries to run the 64-bit version of the application. If the 64-bit version is not available, prints an error and exits with an error.
-bg	Enables the back ground mode.
-cmdfile <i>file</i>	Specifies the command file name. If a file with this name already exists in your unix directory, the new command file will overwrite the existing file. <i>Default:</i> rc.cmd
-ctos	Starts RTL Compiler with a C_to_Silicon_Compiler_L license.
-debug3264	Prints the environment, updated by the wrapper and the command launched.
-display <i>string</i>	Sets the DISPLAY environment variable to the specified value. For example: rc -display cds641:1 If you omit this option and you want to use the graphical user interface, you need to set the DISPLAY environment variable before you start RTL Compiler.
-E	Specifies that RTL Compiler must exit if a script error is found.
-execute <i>command</i>	Specifies the command or Tcl code to execute as a quoted string before any other files specified with the -files option are processed.
-files <i>file</i>	Specifies the names of the scripts (or command files) to execute. To specify multiple files, enclose the list in quotes.
-gui	Starts RTL Compiler with the Graphical User Interface (GUI) visible. Note: GUI commands are only available in the GUI version of RTL Compiler. See Chapter 3, “GUI Text” for detailed information on GUI commands.

Command Reference for Encounter RTL Compiler

General

-logfile *logfile* Specifies the log file name. If a file with this name already exists in your unix directory, the new log file will overwrite the existing file (new content with the same filename).

Default: rc.log

-lsf_cpus *integer* Specifies the number of LSF CPUs to use for super-threading.

-lsf_queue *string* Specifies the name of the LSF queue.

-N *integer* Specifies the number of licenses to use for VDI and RC-L.

Note: When using a Virtuoso Digital Implementation license, you can only stack two licenses, increasing your capacity limit to 100 K instances.

Note: When using RC-L, using one license has a capacity limit of 100K instances for the mapped netlist, using three licenses increases the capacity limit to 300 K instances, while using four licenses, enables unlimited capacity.



The licenses must be on the same server.

-no_custom Specifies to read only the master .synth_init file, located in the installation directory.

By default, RTL Compiler also loads the initialization file in your home directory and in your current design directory.

-nogui Starts RTL Compiler with the Graphical User Interface (GUI) disabled.

-overwrite Allows overwriting of the default and specified command and log files.

-plat *platform* Allows you to override the default platform selection when you launch the tool from the following directory:

install_root/bin

-post *string* Specifies the command to be executed after the file(s) specified with the -files option is (are) processed.

-queue Puts an RTL Compiler session in a queue if a license is not currently available. Once an RTL Compiler license becomes available, an RTL Compiler session is launched.

-quiet3264 Suppresses warning, error, and info messages generated by the -32, -32only, -3264, -64, -64only, or -6432 options.

Command Reference for Encounter RTL Compiler

General

-rcl	Launches RTL Compiler with the RTL_Compiler_L license.
-rcp	Launches RTL Compiler with the RTL_Compiler_Physical license.
-use_license <i>string</i>	<p>Specifies which license to use at startup. If the specified license is unavailable, startup will not continue and the command will fail. When you specify this option multiple times, the command looks for the first available license starting with the first specified one.</p> <p>If no license is specified, RTL Compiler checks out licenses in the following order:</p> <pre>RTL_Compiler_Ultra RTL_Compiler_Physical RTL_Compiler_Verification RTL_Compiler_L Virtuoso_Digital_Implem</pre>
-v3264	Prints the wrapper's version string.
-vdi	Launches RTL Compiler with the Virtuoso Digital Implementation license. It will first try to use the Virtuoso_Digital_Implem license. If that license is unavailable, then it will use the Virtuoso_Digital_Implment license.
-version	Returns the version number without launching the executable.
-wait <i>integer</i>	<p>Specifies the queue wait timeout in seconds.</p> <p>Note: This option only applies if you specified -queue.</p> <p><i>Default:</i> 600</p>

Examples

- The following example first starts RTL Compiler, then executes a script file.

```
unix> rc  
rc:/> include script.g
```

- The following example specifies two script files on the command line:

```
unix> rc -f script1.g -f script2.g
```

- The following example specifies a logfile name with the -logfile option.

```
unix> rc -logfile pov.log
```



Do not use the UNIX `tee` command and pipe (`|`) to specify your logname: doing so would not allow you to use the `control-c` key sequence to gracefully exit a process like incremental optimization.

- The following example uses the `-execute` option to execute a script file:

```
unix> rc -execute "include script.g"
```

- The following example specifies the library path in the command line:

```
unix> rc -ex "set_attribute lib_search_path /net/serverx/libs/mylib"
```

- The following commands start RTL Compiler with the `RTL_Compiler_Physical` license:

```
unix> rc -rcp
```

```
unix> rc -use_license RTL_Compiler_Physical
```

- The following commands have the same effect. Therefore, you should use one or the other and not both in conjunction:

```
unix> rc -use_license Virtuoso_Digital_Implement
```

```
unix> rc -vdi
```

- The following example launches RTL Compiler on four LSF CPUs in the LSF queue named `my_queue`:

```
unix> rc -lsf_cpus 4 -lsf_queue my_queue
```

- The following example will launch not only RTL Compiler, but all Cadence tools in 64-bit mode:

```
unix> setenv CDS_AUTO_64BIT ALL
```

```
unix> rc
```

- The following examples specify that two Virtuoso Digital Implementation licenses be used:

```
unix> rc -vdi -N 2
```

```
unix> rc -N 2 -use_license Virtuoso_Digital_Implement
```

Related Information

Affected by this attribute:

[command_log](#)

redirect

```
redirect [-append] [-tee] [-variable] [-mesg] target command
```

Redirects the standard output to a file or variable. You can write out a gzip compressed file by adding the .gz extension to the filename.

Note: Messages generated by the command whose output you are redirecting, can be mixed with the command output. To limit the number of messages included, you can change the value of the information_level root attribute.

Options and Arguments

-append	Append the generated output to the specified file or Tcl variable.
-mesg	Prevents messages from being suppressed.
-tee	Also writes the output to standard output (<code>stdout</code>) .
-variable	Redirects the output to a Tcl variable.
<i>command</i>	Specifies the command or Tcl code to execute as a quoted string.
<i>target</i>	Specifies the name of the file or variable to which to write the output.

Examples

- The following example sends the output generated by the `report gates` command to a file called `gates.rep`:

```
redirect gates.rep "report gates"
```

- The following example appends information to the existing `gates.rep` file:

```
redirect -append gates.rep "report gates"
```

- The following example sends the `report` to `stdout` and to a file on the disk:

```
redirect -tee gates.rep "report gates"
```

- The following example prevents output generated during reading of the script from being sent to the screen by sending it to `/dev/null`.

```
redirect /dev/null "include script.g"
```

Command Reference for Encounter RTL Compiler

General

- The following example stores the timing report in a variable \$rep_var. You can use Tcl commands to manipulate or parse the variable.

```
redirect -variable rep_var "report timing"
```

- The following examples output a timing and a scan_power gzip compressed report file:

```
redirect file.gz "report timing"
```

```
redirect file.gz "report scan_power"
```

reset_attribute

```
reset_attribute
{ attribute_name [object...] [-quiet]
| -h {attribute_name|*} {type|*}}
```

Resets an attribute to its default value.

You can also use this command to list the attributes whose value can be reset.

Options and Arguments

<i>attribute_name</i>	Specifies the attribute that should be returned to its default value. The "*" wildcard character is supported.
<i>object</i>	Specifies the object for which the attribute values should be returned to the default values.
-quiet	Suppresses those messages that indicate which attributes and objects are being affected.
<i>type</i>	Specifies the object type for which you want the list of attribute names that can be reset. The "*" wildcard character is supported. Check the command help for a list of the valid object types.

Examples

- The following example specifies that all retiming attributes should be returned to their default values:

```
rc:/> reset_attribute retime* *
```
- The following example specifies that all attributes on all sequential instances be returned to their default values:

```
rc:/> reset_attribute * [find / -instance *seq/*]
```
- The following command lists all valid attributes that you can reset:

```
rc:/> reset_attribute -h * *
```

Both *type* and *attribute_name* accept wildcard strings.

resume

resume

Restarts the current Tcl process and brings back the regular `rc:/>` prompt. This command only works in conjunction with the `suspend` command. See the `suspend` command to see how these commands work together to stop and restart a Tcl process.

Related Information

Related commands: [stop_suspend](#) on page 111
[suspend](#) on page 114

sdc_shell

`sdc_shell`

Opens the SDC shell within RTL Compiler. All SDC commands can be used without the `dc::` prefix inside the SDC shell. The command `exit` quits the SDC shell and returns you back to the RTL Compiler prompt.

Example

- The following example launches the SDC shell within RTL Compiler and then exits:

```
rc:/> sdc_shell
Info      : Entering sdc_shell. [SDC-300]
          : All sdc commands will work without the dc::: prefix inside sdc_shell.
          : Type 'exit' to leave the shell.
sdc_shell> exit
Info      : Leaving sdc_shell. [SDC-301]
          : Type sdc_shell to use it again.
rc:/>
```

set_attribute

```
set_attribute
  { attribute_name attribute_value [objects]
    [-quiet] [-lock]
    | -h {attribute_name | *} {type|*} }
```

Sets the value of a specified attribute.

You can also use this command to list

- All attributes associated with a given object type and whose values you can set
- All objects to which a given attribute applies and whose value you can set

Attributes are placed on directory objects to control the way they are processed by RTL Compiler. They can also be used to control the synthesis process, the style of the generated code, and numerous other things. A complete list of all attributes is contained in the [*Attribute Reference for Encounter RTL Compiler*](#).

Note: Not all attributes can be set. Attempting to set a *read-only* attribute returns an error. The *Attribute Reference for Encounter RTL Compiler* indicates whether an attribute is read-write or read-only.

Options and Arguments

<i>attribute_name</i>	Specifies the name of the attribute whose value you want to set.
<i>attribute_value</i>	Specifies the new attribute value. The value can be either a Boolean, integer, or string. A compound string (containing spaces) should be represented as a list using double-quotes or braces.
<i>-lock</i>	Locks the specified attribute's value so that it cannot be changed. The attribute becomes read-only.
<i>objects</i>	Specifies the path(s) to the objects. <i>Default:</i> current directory
<i>-quiet</i>	Suppresses those messages that indicate which objects are being affected. Alternatively, when setting an attribute on an object, an information message will not be printed.

<i>type</i>	Specifies the object type for which you want the list of attribute names. Check the command help for a list of the valid object types.
-------------	--

Examples

- The following example lists all valid attributes that you can set:

```
rc:/> set_attribute * * -help
```

Both *type* and *attribute_name* accept wildcard strings.

- The following example lists all valid attribute names that contain the string *dont*:

```
rc:/> set_attribute *dont* * -help
```

- The following example sets the information_level attribute, which controls the verbosity of the tools, to the value of 5 and assumes the current directory for the path:

```
rc:/> set_attribute information_level 5  
Setting attribute of root /: 'information_level' = 5
```

- In the following example, the path needed to be specified because *information_level* is a root attribute and would not have been found in the current path:

```
rc:/designs/alu> set_attribute information_level 5 /  
Setting attribute of root /: 'information_level' = 5
```

- The following locks the technology library search path to /home/Test/foo by locking the *lib_search_path* attribute. For the rest of the session, the *lib_search_path* attribute becomes read-only.:

```
rc:/> set_attribute -lock lib_search_path /home/Test/foo
```

Related Information

Affects these commands:

[ls](#) on page 50

[get_attribute](#) on page 77

shell

`shell command_string`

Executes a UNIX shell command from the RTL Compiler shell.

When executing shell commands, RTL Compiler uses /bin/sh.



Attempting to change the working directory for the RTL Compiler shell using `shell cd ..` is not possible because each `shell` command is executed in its own shell, and that shell is killed once the command is complete.

Options and Arguments

<code>command_string</code>	Specifies the UNIX command to execute. You can specify any valid UNIX command. A sequence of commands must be specified in the same string.
-----------------------------	--

Examples

- The following example uses the `sh` command to get the current date:

```
rc:/> shell date  
...
```

Related Information

Affected by this command:	lcd on page 83
Related commands:	lls on page 90 lpwd on page 93

stop_suspend

`stop_suspend`

Terminates the script execution when you have stopped the current Tcl process with the `suspend` command (which brings up the `rc-suspend:/>` prompt) and returns from the `suspend` command loop (brings back the regular `rc:/>` prompt).

This allows you to have access to the GUI to analyze any issues reported so far.

Related Information

Related commands: [resume](#) on page 106

[suspend](#) on page 114

string_representation

```
string_representation string
```

Converts a Tcl object into a string.

Options and Arguments

<i>string</i>	Specifies the object to be converted.
---------------	---------------------------------------

Example

The following example shows the effect of manipulations on a Tcl list (list of object pointers) versus a string.

In the first part of the example, you create a variable `list` that contains the pointers to the instances in the design that have `sub` in their name. The tool finds two instances. Next you remove one of the instances and print the content of the variable `list` again. Because the variable contains a list of pointers, the pointer to the second object is now replaced by `object_deleted`. When you rename the first object in the list and then print the content of the variable `list` again, the tool returns the pointer to the renamed instance and a pointer indicating that one object was deleted.

```
rc:/> set list [find / -inst *sub*]
rc:/> /designs/top/U1/instances_hier/sub1
      /designs/top/U2/instances_hier/sub2
rc:/> rm [find U2 -instance sub2]
rc:/> puts $list
rc:/> /designs/top/U1/instances_hier/sub1 object_deleted
rc:/> mv /designs/top/U1/instances_hier/sub1 mySub1
rc:/> puts $list
rc:/> /designs/top/U1/instances_hier/mySub1 object_deleted
```

If you convert the Tcl list into a string before you perform all these manipulations, the content of the variable `list` will not be affected by the manipulations.

```
rc:/> set list [string_representation [find / -inst *sub*]]
rc:/> { /designs/top/U1/instances_hier/sub1
      /designs/top/U2/instances_hier/sub2 }
rc:/> rm [find U2 -instance sub2]
rc:/> mv /designs/top/U1/instances_hier/sub1 mySub1
rc:/> puts $list
rc:/> { /designs/top/U1/instances_hier/sub1
      /designs/top/U2/instances_hier/sub2 }
```

suppress_messages

```
suppress_messages [-n integer] [-quiet] message_id...
```

Suppresses printing of the specified message in the log file.

Options and Arguments

<i>message_id</i>	Specifies the message identification.
-n <i>integer</i>	Prints the specified message only <i>n</i> number of times. The default value is 0.
-quiet	Suppresses those messages that indicate which attributes are being affected for the messages that are being suppressed.

Examples

- The following example suppresses the VLOG-1 and VLOG-2 messages:

```
rc:/> suppress_messages { VLOG-1 VLOG-2 }
```

- The following example shows the effect of the -quiet option:

with the -quiet option:

```
Warning : Potential variable name conflict. [TUI-666]
          : A variable that controls the tool's behavior was set. The
          'lbr_use_test_cell_seq_mux_scan' variable has the same name as an
          internal variable.
          Setting attribute of root '//': 'lib_search_path' =
          /home/rchap/reg/rc/dft/SMG/native/analysis/test2/LIB
```

without the -quiet option:

```
Warning : Potential variable name conflict. [TUI-666]
          : A variable that controls the tool's behavior was set. The
          'lbr_use_test_cell_seq_mux_scan' variable has the same name as an
          internal variable.
          Setting attribute of message 'LBR-41': 'max_print' = 0
          Setting attribute of message 'LBR-146': 'max_print' = 0
          Setting attribute of message 'LBR-23': 'max_print' = 0
          Setting attribute of root '//': 'lib_search_path' =
          /home/rchap/reg/rc/dft/SMG/native/analysis/test2/LIB
```

Related Information

Related command: [unsuppress_messages](#) on page 118

suspend

`suspend`

Stops the current Tcl process and brings up the `rc-suspend : />` prompt. Any commands or attributes that are issued during this time will not have access to the temporary variables from the suspended Tcl process. However, global variables and Tcl processes can still be accessed. Type `resume` to restart the process from where it was stopped.

Note: Any commands that you enter after issuing the `suspend` command but before the `resume` command will not be included in Tcl's command history. However, these commands will be included in RTL Compiler's command editor history.

Related Information

Related commands: [resume on page 106](#)
[stop_suspend on page 111](#)

tcl_load

```
tcl_load fileName [packageName [interp]]
```

This command loads binary code from a file into the application's address space and calls an initialization procedure in the package to incorporate it into an interpreter.

Note: The Tcl command `load` must be renamed to `tcl_load` to avoid conflict with the RTL Compiler compiler `load` which is aliased to `read_hdl`.

Options and Arguments

<i>filename</i>	Is the name of the file containing the code. Its exact form varies from system to system but on most systems it is a shared library, such as a .so file under Solaris.
<i>packageName</i>	Is the name of the package, and is used to compute the name of an initialization procedure.
<i>interp</i>	Is the path name of the interpreter into which to load the package (see the <code>interp</code> manual entry for details). If this argument is omitted, it defaults to the interpreter in which the <code>load</code> (<code>tcl_load</code>) command was invoked.

Description

Once the file has been loaded into the application's address space, one of two initialization procedures will be invoked in the new code. Typically the initialization procedure will add new commands to a Tcl interpreter. The name of the initialization procedure is determined by *packageName* and whether or not the target interpreter is a safe one. For normal interpreters the name of the initialization procedure will have the form `pkg_Init`, where *pkg* is the same as *packageName* except that the first letter is converted to upper case and all other letters are converted to lower case. For example, if *packageName* is `foo` or `FOO`, the initialization procedure's name will be `Foo_Init`.

If the target interpreter is a safe interpreter, then the name of the initialization procedure will be `pkg_SafeInit` instead of `pkg_Init`. The `pkg_SafeInit` function should be written carefully, so that it initializes the safe interpreter only with partial functionality provided by the package that is safe for use by untrusted code. For more information on Safe-Tcl, see the `safe` manual entry.

Command Reference for Encounter RTL Compiler

General

The initialization procedure must match the following prototype:

```
typedef int Tcl_PackageInitProc(Tcl_Interp *interp);
```

The `interp` argument identifies the interpreter in which the package is to be loaded. The initialization procedure must return `TCL_OK` or `TCL_ERROR` to indicate whether or not it completed successfully; in the event of an error it should set the interpreter's result to point to an error message. The result of the `load` command will be the result returned by the initialization procedure.

The actual loading of a file will only be done once for each `fileName` in an application. If a given `fileName` is loaded into multiple interpreters, then the first load will load the code and call the initialization procedure; subsequent loads will call the initialization procedure without loading the code again. It is not possible to unload or reload a package.

The `load` command also supports packages that are statically linked with the application, if those packages have been registered by calling the `Tcl_StaticPackage` procedure. If `fileName` is an empty string, then `packageName` must be specified.

If `packageName` is omitted or specified as an empty string, Tcl tries to guess the name of the package. This may be done differently on different platforms. The default guess, which is used on most UNIX platforms, is to take the last element of `fileName`, strip off the first three characters if they are lib, and use any following alphabetic and underline characters as the module name. For example, the command `load libxyz4.2.so` uses the module name xyz and the command `load bin/last.so {}` uses the module name last.

If `fileName` is an empty string, then `packageName` must be specified. The `load` command first searches for a statically loaded package (one that has been registered by calling the `Tcl_StaticPackage` procedure) by that name; if one is found, it is used. Otherwise, the `load` command searches for a dynamically loaded package by that name, and uses it if it is found. If several different files have been loaded with different versions of the package, Tcl picks the file that was loaded first.

Bugs

If the same file is loaded by different fileNames, it will be loaded into the process's address space multiple times. The behavior of this varies from system to system (some systems may detect the redundant loads, others may not).

Example

The following is a minimal extension:

```
#include <tcl.h>
#include <stdio.h>
static int fooCmd(ClientData clientData,
    Tcl_Interp *interp, int objc, char * CONST objv[]) {
    printf("called with %d arguments\n", objc);
    return TCL_OK;
}
int Foo_Init(Tcl_Interp *interp) {
    if (Tcl_InitStubs(interp, "8.1", 0) == NULL) {
        return TCL_ERROR;
    }
    printf("creating foo command");
    Tcl_CreateObjCommand(interp, "foo", fooCmd, NULL, NULL);
    return TCL_OK;
}
```

When built into a shared/dynamic library with a suitable name (e.g. libfoo.so on Solaris and Linux) it can then be loaded into Tcl with the following:

```
# Load the extension
switch $tcl_platform(platform) {
    unix {
        load ./libfoo[info sharedlibextension]
    }
}

# Now execute the command defined by the extension
200
```

See Also

[info sharedlibextension](#), [Tcl_StaticPackage\(3\)](#), [safe\(n\)](#)

Keywords

binary code, loading, safe interpreter, shared library

unsuppress_messages

`unsuppress_messages message_id...`

Allows a previously suppressed message to be printed.

Options and Arguments

message_id Specifies the message identification.

Examples

- The following example suppresses the VLOG-1 and VLOG-2 messages and then allows them to be printed again:

```
rc:/> suppress_messages { VLOG-1 VLOG-2 }
rc:/> unsuppress_messages { VLOG-1 VLOG-2 }
```

Related Information

Related command: [suppress_messages](#) on page 113

GUI Text

- [General GUI Text Commands](#) on page 120
- [HDL Viewer GUI Text Commands](#) on page 126
- [Schematic Viewer GUI Text Commands](#) on page 129
- [Physical Viewer GUI Text Commands](#) on page 134

General GUI Text Commands

- [gui_balloon_info](#) on page 121
- [gui_hide](#) on page 121
- [gui_info](#) on page 121
- [gui_legend](#) on page 121
- [gui_raise](#) on page 122
- [gui_reset](#) on page 123
- [gui_resume](#) on page 123
- [gui_selection](#) on page 123
- [gui_show](#) on page 124
- [gui_status](#) on page 124
- [gui_suspend](#) on page 124
- [gui_update](#) on page 125

gui_balloon_info

`gui_balloon_info`

Retrieves the text from the last balloon info.

gui_hide

`gui_hide`

Hides the GUI. Type the `gui_show` command to re-display the GUI.

Related Information

Related commands: [gui_raise](#) on page 122

[gui_show](#) on page 124

gui_info

`gui_info string`

Adds the specified text string in the info section of the status bar. The text remains until it is overwritten.

This command is usually used to print persistent messages (for example, Design is mapped).

gui_legend

`gui_legend -title titlename [-cancel string] string`

Creates a GUI legend. This is useful if you need a simple legend to color code information.

Note: You can use this command with the Physical Viewer and Schematic Viewer.

Options and Arguments

`-cancel string` Specifies a Tcl proc that can be run when the Close button is pressed.

`string` Specifies the data set to display in the legend.

`-title titlename` Specifies the legend dialog title.

Example

- The following commands show how to create a legend.

The first set of commands create the data set for the legend:

```
set entry1 [list red "This is entry 1"]
set entry2 [list green "This is entry 2"]
set entry3 [list blue "This is entry 3"]
set data [list $entry1 $entry2 $entry3]
```

The next command specifies the TCI procedure to run when the Close button is pressed.

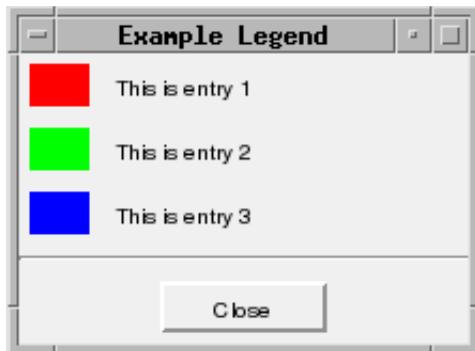
```
proc foo {} {
    puts "Legend cancel"
}
```

The last command specifies the data set and title to display in the legend, and the procedure to run when the Close button is pressed:

```
gui_legend -title "Legend Example" -cancel foo $data
```

The following figure shows the resulting GUI legend.

Figure 3-1 GUI Legend



gui_raise

```
gui_raise [-nosync]
```

Keeps the GUI window on top of all other windows.

Options and Arguments

-nosync

Does not synchronize the GUI when the GUI is raised.

Related Information

Related commands: [gui_hide](#) on page 121
[gui_show](#) on page 124

gui_reset

`gui_reset`

Resets the busy indicator if it remains busy.

The busy indicator can remain red if the script that set the busy indicator is broken and therefore the busy indicator does not get reset or cleared.

gui_resume

`gui_resume`

Resumes automatic GUI updates.

Related Information

Related command: [gui_suspend](#) on page 124

gui_selection

`gui_selection`

Returns the selection list, which consists of the object path for instance, net, pin, and port objects. The list can contain multiple objects.

Example

- The following example returns a combinational instance:

```
rc:/> gui_selection  
/designs/mul_clk/instances_comb/g14
```

If you enable the toolbar by un-checking *Hide Toolbar* under the Preferences menu, the complete path for the last selected object will be displayed.

gui_show

`gui_show [-nosync]`

Displays the GUI after using `gui_hide` command.

Options and Arguments

`-nosync` Does not synchronize the GUI when the GUI is displayed.

Related Information

Related commands: [gui_hide](#) on page 121

[gui_raise](#) on page 122

gui_status

`gui_status string`

Adds the specified text string in the status section of the status bar (window right to the busy indicator). The text remains until it is overwritten.

This command is usually used to print transient messages (for example, Loading library *name*).

gui_suspend

`gui_suspend`

Suspends the automatic GUI updates.

Related Information

Related command: [gui_resume](#) on page 123

gui_update

`gui_update`

Updates the GUI. This is the same as selecting *Update GUI* from the File menu.

Related Information

Related commands:

<code>gui_resume</code> on page 123
<code>gui_suspend</code> on page 124

HDL Viewer GUI Text Commands

- [gui_hv_clear](#) on page 127
- [gui_hv_get_file](#) on page 127
- [gui_hv_load_file](#) on page 127
- [gui_hv_set_indicators](#) on page 128

gui_hv_clear

`gui_hv_clear`

Removes all the data in the HDL Viewer.

gui_hv_get_file

`gui_hv_get_file`

Returns the name of the file currently loaded in HDL Viewer.

gui_hv_load_file

`gui_hv_load_file filename`

Loads the specified file name into the HDL Viewer. Same as clicking the *Open HDL File* icon in the HDL Viewer.

gui_hv_set_indicators

```
gui_hv_set_indicators [-clear] line_number column_number
```

Sets the *line* and *column* number indicators. Using this command is useful if you are writing a script and you want to highlight a specific line in the HDL Viewer.

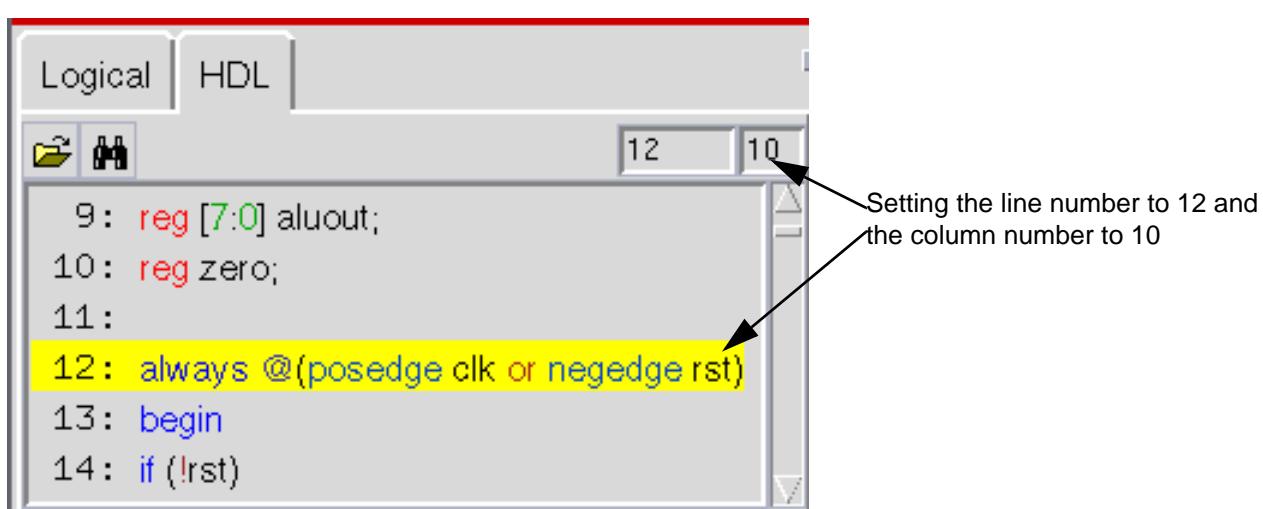
Options and Arguments

<i>-clear</i>	Removes all tag highlighting. If this option is not used, then the specified line number is highlighted.
<i>column_number</i>	Specifies the column number. If the <i>column_number</i> argument is not specified, then the default is 0.
<i>line_number</i>	Specifies the line number.

Example

- The following command highlights the specified line number and sets the line number to 12 and the column number to 10, as shown in Figure 3-2.
`rc:/> gui_hv_set_indicators 12 10`
- The following commands sets the line number to 12, the column number to 10, and removes the highlighting:
`rc:/> gui_hv_set_indicators 12 10 -clear`

Figure 3-2 Setting Line and Column Numbers in the HDL Viewer



Schematic Viewer GUI Text Commands

- [gui_sv_clear](#) on page 130
- [gui_sv_cone](#) on page 130
- [gui_sv_get_instance](#) on page 130
- [gui_sv_grey](#) on page 130
- [gui_sv_highlight](#) on page 131
- [gui_sv_load](#) on page 132
- [gui_sv_snapshot](#) on page 133
- [gui_sv_toolbar_button](#) on page 133

gui_sv_clear

`gui_sv_clear`

Clears highlighting and selection lists in the Schematic Viewer.

gui_sv_cone

`gui_sv_cone instance`

Displays the specified instance into a new schematic cone viewer.

Note: This command is the text -equivalent for *Open in —Schematic Viewer (cone)* command in the Schematic Viewer, which is available when an instance is selected.

Options and Arguments

<code>instance</code>	Specifies the instance you want to start drawing a cone from (the instance that you would have highlighted in the schematic before bringing up the context-sensitive menu).
-----------------------	---

gui_sv_get_instance

`gui_sv_get_instance`

Returns the objects for the currently displayed hierarchical instance.

gui_sv_grey

`gui_sv_grey [on | off]`

Controls the grey mode. You can also right-click the mouse button in the Schematic Viewer and select *Grey Mode On* or *Grey Mode Off*.

Options and Arguments

<code>-on</code>	Turns on grey mode in the Schematic Viewer.
<code>-off</code>	Turns off grey mode in the Schematic Viewer.

gui_sv_highlight

```
gui_sv_highlight [-append] [-color color] [-center]
                 [-from {port|pin}] [-to {port|pin}] [-zoomto]
                 {port|pin|net|instance}
```

Highlights the specified object in the Schematic Viewer.

Options and Arguments

-append	Appends an object to the highlighted list.
-center	Centers an object in the Schematic Viewer.
-color <i>color</i>	Specifies the color for highlighting an object.
-from { <i>port pin</i> }	Specifies the start point of the highlighted net.
-to { <i>port pin</i> }	Specifies the end point of the highlighted net.
-zoomto	Zooms into the specified port, pin, instance or net.

Examples

- The following Tcl procedure highlights inverters in the Schematic Viewer

```
# highlight inverters
proc highlight_inverters {idir type odir} {
    # only proceed if no object under cursor
    if {$type != "none"} return
    # clear any highlight and selection
    gui_sv_clear
    _find_inverters $idir
}
proc _find_inverters {idir} {
    # search for inverters
    foreach inst [find $idir -maxdepth 2 -instance *comb/*] {
        if {[get_attribute inverter $inst] == "true"} {
            gui_sv_highlight $inst -append -color green
        }
    }
}
```

Command Reference for Encounter RTL Compiler

GUI Text

- The following example highlights the falling clock edge flip-flops

```
# highlight falling clock edge flip-flops
proc highlight_falling_edge_ff {idir type odir} {
    # only proceed if no object under cursor
    if {$type != "none"} return
    # clear any highlight and selection
    gui_sv_clear
    _find_falling_edge_ff $idir
}

proc _find_falling_edge_ff {idir} {
    # search for sequential gates with falling clock edge
    foreach inst [find $idir -maxdepth 2 -instance *seq/*] {
        set libcell [get_attribute libcell $inst]
        if {$libcell == ""} continue
        if {[get_attribute flop $libcell] == "true"} {
            foreach libpin [find $libcell -libpin *] {
                if {[get_attribute output $libpin] == "true"} {
                    foreach arc [get_attribute incoming_timing_arcs $libpin] {
                        set liberty [get_attribute liberty_attributes $arc]
                        if {[lsearch $liberty falling_edge] != -1} {
                            gui_sv_highlight $inst -append
                        }
                    }
                }
            }
        }
    }
}
```

gui_sv_load

```
gui_sv_load {design | instance}
```

Specifies an hierarchical instance or design to load into the main Schematic Viewer.

gui_sv_snapshot

```
gui_sv_snapshot  
    [-overwrite] [-no_fit] [-ps] file
```

Saves a snapshot of the part of the design that is visible in the Schematic Viewer.

Options and Arguments

<i>file</i>	Specifies the name of the file in which to save the snapshot.
-no_fit	Specifies to not fit the snapshot to the full screen.
-overwrite	Specifies to overwrite an existing file. If you omit this option, you'll get a message that indicates that the file exists.
-ps	Specifies to create the file in PostScript format. If this option is omitted, a GIF file will be created.

gui_sv_toolbar_button

```
gui_sv_toolbar_button  
    -icon string -proc string -tip string
```

Adds a customized button to the toolbar of the schematic viewer.

Options and Arguments

<i>-icon string</i>	Specifies the GIF or PNM file that contains the drawing of the icon. The icon size is 24x24 and has a gray94 background.
<i>-proc string</i>	Specifies the name of the Tcl procedure to execute. The procedure must be loaded before you press the button to call it.
<i>-tip string</i>	Specifies the tool tip to be displayed.

Physical Viewer GUI Text Commands

- [gui_pv_airline_add](#) on page 136
- [gui_pv_airline_add_custom](#) on page 137
- [gui_pv_airline_delete](#) on page 138
- [gui_pv_airline_display](#) on page 138
- [gui_pv_airline_raw_add](#) on page 139
- [gui_pv_airline_raw_add_custom](#) on page 140
- [gui_pv_clear](#) on page 141
- [gui_pv_connectivity_airlines](#) on page 141
- [gui_pv_deselect](#) on page 141
- [gui_pv_display_collection](#) on page 143
- [gui_pv_draw_box](#) on page 144
- [gui_pv_draw_circle](#) on page 145
- [gui_pv_draw_line](#) on page 146
- [gui_pv_draw_triangle](#) on page 147
- [gui_pv_highlight](#) on page 148
- [gui_pv_highlight_update](#) on page 150
- [gui_pv_label](#) on page 150
- [gui_pv_preferences](#) on page 151
- [gui_pv_redraw](#) on page 151
- [gui_pv_select](#) on page 151
- [gui_pv_selection](#) on page 153
- [gui_pv_snapshot](#) on page 153
- [gui_pv_steiner_tree](#) on page 155
- [gui_pv_update](#) on page 155
- [gui_pv_zoom_box](#) on page 156
- [gui_pv_zoom_fit](#) on page 157

Command Reference for Encounter RTL Compiler

GUI Text

- [gui_pv_zoom_in](#) on page 157
- [gui_pv_zoom_out](#) on page 157
- [gui_pv_zoom_to](#) on page 157

gui_pv_airline_add

```
gui_pv_airline_add -name name  
    -from {port|instance} -to {port|instance}  
    [-color color] [-label label] [-nodisplay]
```

Adds the specified airline to the Physical Viewer.

Options and Arguments

-color color	Specifies the airline color. <i>Default:</i> blue
-from {port instance}	Specifies the from object.
-label label	Specifies a label to place at the center of the airline.
-name name	Specifies the airline name.
-nodisplay	Specifies not to display the airline.
-to {port instance}	Draws an airline from the center of the from object to the center of the to object.

Example

- The following command creates a red airline named test from the g442 port to the g412 port with the airline label, as shown in Figure 3-3.

```
gui_pv_airline_add -from g442 -to g412 -color red -label airline -name test
```

Figure 3-3 Specified Airline in the Physical Viewer



gui_pv_airline_add_custom

```
gui_pv_airline_add_custom -name name
    -from {port|instance} -to {port|instance}
    [-color color] [-label label | -nolabel]
    [-nodisplay] [-noglyph] [-nomiddle]
    [-type {0|1|2}] [-width integer]
```

Adds a customized airline between two objects in the Physical Viewer.

Options and Arguments

-color <i>color</i>	Specifies the airline color. <i>Default:</i> blue
-from { <i>instance</i> <i>port</i> }	Specifies the from object.
-label <i>label</i>	Specifies a label to place at the center of the airline.
-name <i>name</i>	Specifies the airline name.
-nodisplay	Specifies not to display the airline at the time you create it.
-noglyph	Specifies not to display the airline glyphs.
-nolabel	Specifies not to display the airline label.
-nomiddle	Specifies not to display the airline arrow in the middle.
-to { <i>instance</i> <i>port</i> }	Draws an airline from the center of the from object to the center of the to object.
-type {0 1 2}	Specifies the airline glyph type at the endpoints of the airline. 0—no glyphs are added 1—arrow type glyphs are added 2—circle type glyphs are added
-width <i>integer</i>	Specifies the airline width.

gui_pv_airline_delete

`gui_pv_airline_delete name`

Deletes airlines from the Physical Viewer with the specified *name*.

Options and Arguments

name

Specifies the airline name to be deleted.

The airline name must have been added using the
[gui_pv_airline_add](#) command.

Example

- The following command removes the airline `test` from the Physical Viewer:

`rc:/> gui_pv_airline_delete test`

gui_pv_airline_display

`gui_pv_airline_display name`

Options and Arguments

name

Specifies the airline name to be displayed.

The airline name must have been added using the
[gui_pv_airline_add](#) command.

Displays the specified airline name in the Physical Viewer.

Example

- The following command removes the airline `test` from the Physical Viewer:

`rc:/> gui_pv_airline_display test`

gui_pv_airline_raw_add

```
gui_pv_airline_raw_add [-name name]
    [-color color] [-label label]
    [-nodisplay] -fx float -fy float
    -tx float -ty float
```

Creates an airline in the Physical Viewer between two points specified by their coordinates.

Options and Arguments

<code>-color color</code>	Specifies the airline color. <i>Default:</i> blue
<code>-fx float</code>	Specifies the x coordinate of the from object.
<code>-fy float</code>	Specifies the y coordinate of the from object.
<code>-label label</code>	Specifies a label to place at the center of the airline.
<code>-name name</code>	Specifies the airline name.
<code>-nodisplay</code>	Specifies not to display the airline.
<code>-tx float</code>	Specifies the x coordinate of the to object.
<code>-ty float</code>	Specifies the y coordinate of the to object.

gui_pv_airline_raw_add_custom

```
gui_pv_airline_raw_add_custom -name name
    -fx float -fy float -tx float -ty float
    [-color color] [-label label | -nolabel]
    [-nodisplay] [-noglyph] [-nomiddle]
    [-type {0|1|2}] [-width integer]
```

Creates a customized airline in the Physical Viewer between two points specified by their coordinates.

Options and Arguments

<code>-color color</code>	Specifies the airline color. <i>Default:</i> blue
<code>-fx float</code>	Specifies the X coordinate of the from object.
<code>-fy float</code>	Specifies the Y coordinate of the from object.
<code>-label label</code>	Specifies a label to place at the center of the airline.
<code>-name name</code>	Specifies the airline name.
<code>-nodisplay</code>	Specifies not to display the airline at the time you create it.
<code>-noglyph</code>	Specifies not to display the airline glyphs.
<code>-nolabel</code>	Specifies not to display the airline label.
<code>-nomiddle</code>	Specifies not to display the airline arrow in the middle.
<code>-tx float</code>	Specifies the X coordinate of the to object.
<code>-ty float</code>	Specifies the Y coordinate of the to object.
<code>-type {0 1 2}</code>	Specifies the airline glyph type at the endpoints of the airline. 0—no glyphs are added 1—arrow type glyphs are added 2—circle type glyphs are added
<code>-width integer</code>	Specifies the airline width.

gui_pv_clear

```
gui_pv_clear [-object_list]
```

Clears highlighting from the specified object list.

Options and Arguments

-object_list Clears all items in the specified list.

gui_pv_connectivity_airlines

```
gui_pv_connectivity_airlines  
  [-append] {instance | port}...
```

Shows the connectivity airlines from the specified instance or port to the instances and ports it is connected to.

Options and Arguments

-append Adds the new airlines to the existing ones.

{instance | port} Specifies the name of the instance or port from where to draw the connectivity airlines.

gui_pv_deselect

```
gui_pv_deselect object...
```

Specifies the list of object types to deselect in the physical viewer.

You can specify any of the following:

blackbox	blockage	bump
cluster	def_pin	gcell
instance	pcell	pdomain
pin	pinstance	port

Command Reference for Encounter RTL Compiler

GUI Text

region

row

gui_pv_display_collection

`gui_pv_display_collection [-group group]`

Displays a collection of objects in the Physical Viewer.

Options and Arguments

`-group group` Specifies the name of the group to be displayed.

A group can be created and appended to by any `gui_pv_xx` command that has the `-collect` and `-group` options.

Related Information

Related commands:	gui_pv_draw_box on page 144 gui_pv_draw_circle on page 145 gui_pv_draw_line on page 146 gui_pv_draw_triangle on page 147 gui_pv_highlight on page 148
-------------------	---

gui_pv_draw_box

```
gui_pv_draw_box -llx float -lly float  
    -width float -height float  
    [-color color] [-collect] [-append] [-stipple]  
    [-group name] [-label label] [-dbu]
```

Draws a box on top of the physical view. You can use this command to annotate information.

Options and Arguments

-append	Appends the object to highlight.
-collect	Collects objects to highlight.
-color <i>color</i>	Specifies the highlight color. <i>Default:</i> red
-dbu	Specifies the coordinates and dimensions in DB units, where one DB unit is 1/1000 micrometer.
-label <i>label</i>	Specifies a label to place at the center of the box.
-group <i>name</i>	Displays only those objects in the specified group. These objects were previously marked for display through the <code>gui_pv_highlight -collect</code> command.
-height <i>float</i>	Specifies the height of the box in micrometer.
-llx <i>float</i>	Specifies the x coordinate of the origin of the box in micrometer.
-lly <i>float</i>	Specifies the y coordinate of the origin of the box in micrometer.
-label <i>string</i>	Specifies the object label.
-stipple	Specifies to use stipple fill pattern to highlight the object.
-width <i>float</i>	Specifies the width of the box micrometer.

gui_pv_draw_circle

```
gui_pv_draw_circle -cx float -cy float  
    -radius float  
    [-color color] [-collect] [-append] [-stipple]  
    [-group name] [-label label] [-dbu]
```

Draws a circle on top of the physical view. You can use this command to annotate information.

Options and Arguments

<code>-append</code>	Appends the object to highlight.
<code>-collect</code>	Collects objects to highlight.
<code>-color color</code>	Specifies the highlight color. <i>Default:</i> red
<code>-cx float</code>	Specifies the x coordinate of the center of the circle in micrometer.
<code>-cy float</code>	Specifies the y coordinate of the center of the circle in micrometer.
<code>-dbu</code>	Specifies the coordinates and dimensions in DB units, where one DB unit is 1/1000 micrometer.
<code>-label label</code>	Specifies a label to place at the center of the box.
<code>-group name</code>	Displays only those objects in the specified group. These objects were previously marked for display through the <code>gui_pv_highlight -collect</code> command.
<code>-label string</code>	Specifies the object label.
<code>-radius float</code>	Specifies the radius of the circle in micrometer.
<code>-stipple</code>	Specifies to use stipple fill pattern to highlight the objects.

gui_pv_draw_line

```
gui_pv_draw_circle -fx float -fy float  
    -tx float -ty float  
    [-color color] [-collect] [-append] [-stipple]  
    [-group name] [-label label] [-dbu]
```

Draws a line between the specified coordinates on top of the physical view. You can use this command to annotate information.

Options and Arguments

<code>-append</code>	Appends the object to highlight.
<code>-collect</code>	Collects objects to highlight.
<code>-color color</code>	Specifies the highlight color. <i>Default:</i> red
<code>-dbu</code>	Specifies the coordinates and dimensions in DB units, where one DB unit is 1/1000 micrometer.
<code>-fx float</code>	Specifies the x coordinate of the first (or from) point in micrometer.
<code>-fy float</code>	Specifies the y coordinate of the first (or from) point in micrometer.
<code>-group name</code>	Displays only those objects in the specified group. These objects were previously marked for display through the <code>gui_pv_highlight -collect</code> command.
<code>-label label</code>	Specifies a label to place at the center of the line.
<code>-stipple</code>	Specifies to use stipple fill pattern to highlight the objects.
<code>-tx float</code>	Specifies the x coordinate of the second (or to) point in micrometer.
<code>-ty float</code>	Specifies the y coordinate of the second (or to) point in micrometer.

Note: One DB unit is 1/1000 micron.

gui_pv_draw_triangle

```
gui_pv_draw_triangle -llx float -lly float  
-width float -height float  
[-color color] [-collect] [-append] [-stipple]  
[-group name] [-label label] [-dbu]
```

Draws an isosceles triangle on top of the physical view with the specified width and height that originates in the specified lower left coordinates. You can use this to annotate information.

Options and Arguments

-append	Appends the object to highlight.
-collect	Collects objects to highlight.
-color <i>color</i>	Specifies the highlight color. <i>Default:</i> red
-dbu	Specifies the coordinates and dimensions in DB units, where one DB unit is 1/1000 micrometer.
-group <i>name</i>	Displays only those objects in the specified group. These objects were previously marked for display through the <i>gui_pv_highlight -collect</i> command.
-height <i>float</i>	Specifies the height of the triangle in micrometer.
-label <i>string</i>	Specifies a label to place at the center of the triangle.
-llx <i>float</i>	Specifies the x coordinate of the lower left point in micrometer.
-lly <i>float</i>	Specifies the y coordinate of the lower left point in micrometer.
-stipple	Specifies to use stipple fill pattern to highlight the objects.
-width <i>float</i>	Specifies the width of the triangle in micrometer.

gui_pv_highlight

```
gui_pv_highlight [-color color] [-collect] [-append]
[-group name] [-label string] [-stipple]
{blockage | gcell | instance | pcell |
pdomain | pin | port | region | row}...
```

Highlights objects in the Physical Viewer

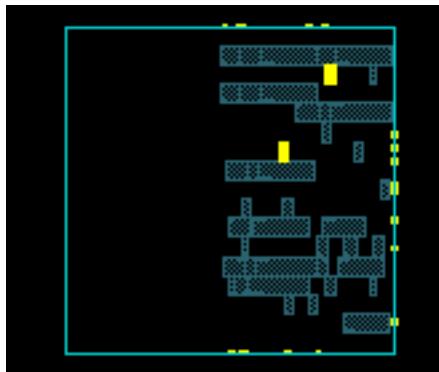
Options and Arguments

-append	Appends the object to highlight.
-collect	Collects objects to highlight
-color <i>color</i>	Specifies the color for highlighting. <i>Default:</i> red
{blockage gcell instance pcell pdomain pin port region row}	Specifies the object to highlight if it is in the scope of the current Physical Viewer.
-group <i>name</i>	Specifies the group name for the object.
-label <i>string</i>	Specifies the object label.
-stipple	Specifies to use stipple fill pattern to highlight the object.

Examples

- The following command highlights the g411 port in yellow (see [Figure 3-4](#) on page 149):
rc:/> gui_pv_highlight -color yellow g411
- The following command adds the g405 port to the highlighted list (see [Figure 3-4](#) on page 149):
rc:/> gui_pv_highlight -color yellow -append g405

Figure 3-4 Highlighting an Object in the Physical Viewer



- The following command specifies the instance `evel` to be highlighted with the `gui_pv_draw_collection` command. The `-group` option indicates that to which group `evel` should belong. In this case, it is `laurence`:

```
rc:/> gui_pv_highlight -collect /designs/bree/instances_hier/evel/ \
    -group laurence
```

Related Information

Related command: [gui_pv_clear](#) on page 141

gui_pv_highlight_update

```
gui_pv_highlight_update -property string
    -value string [-group string]
    {blockage | gcell | instance | pcell | port | region}...
```

Updates the object highlight in the Physical Viewer.

Options and Arguments

{blockage | gcell | instance | pcell | port | region}

Specifies the names of the objects for which to update the highlight.

-group *string* Specifies the object group name.

-property *value* Specifies the property to update.

You can update the colors, the labels, or the stipple pattern.

-value *value* Specifies the object property value.

gui_pv_label

```
gui_pv_label
    [-color color] -x float -y float label
```

Adds a text label at the specified point in the Physical Viewer.

Options and Arguments

-color *color* Specifies the label color.

Default: white.

label Specifies the label name.

-x *float* Specifies the x coordinate for the label.

-y *float* Specifies the y coordinate for the label.

gui_pv_preferences

```
gui_pv_preferences [-name] [-selectable] [-visible]
                   [-color string] [-style {outline | fill | stipple}]
string
```

Configures the preferences for the specified object type in the Physical Viewer.

Options and Arguments

-color <i>string</i>	Specifies the color of the objects of the specified type.
-name	Specifies to display the name of the objects of the specified type.
-selectable	Specifies that objects of the specified type are selectable.
<i>string</i>	Specifies the object type for which to configure the preferences. You can specify any of the following: blackbox, blockage_placement, blockage_derived_placement, blockage_partial_placement, blockage_soft_placement, blockage_routing, bump, cluster, core, die, fence, gcell, guide, macro_cover, macro_fixed, macro_generic, macro_placed, pcell, pin, port, power_domain, region, and row.
-style	Specifies the drawing style for the object type. <i>Default:</i> outline.
-visible	Specifies that the objects of the specified type must be visible.

gui_pv_redraw

```
gui_pv_redraw
```

Redraws the contents of the Physical Viewer.

gui_pv_select

```
gui_pv_select object...
```

Specifies the list of object types to select in the physical viewer.

Command Reference for Encounter RTL Compiler

GUI Text

You can specify any of the following:

blackbox	blockage	bump
cluster	def_pin	gcell
instance	pcell	pdomain
pin	pinstance	port
region	row	

gui_pv_selection

```
gui_pv_selection
```

Returns the list of selected objects in the Physical Viewer.

Note: You can select multiple objects by pressing the Shift key while drawing a region with the left mouse button. All objects overlapping that region are selected.

gui_pv_snapshot

```
gui_pv_snapshot  
[-congestion] [-pin_density] [-utilization]  
[-no_fit] [-overwrite] [-ps] file
```

Saves a snapshot of the part of the design that is visible in the Physical Viewer.

Options and Arguments

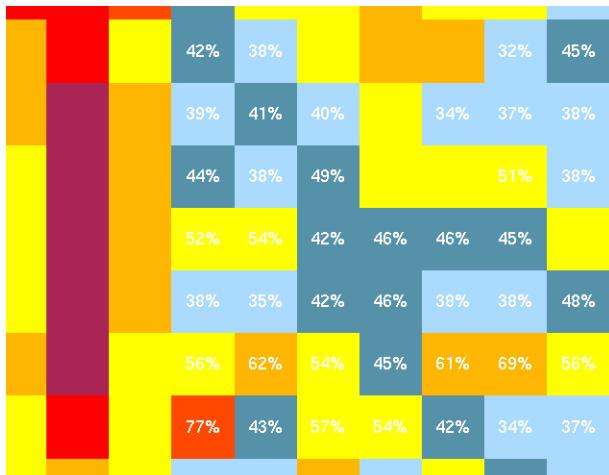
-congestion	Overlays a congestion map on top of what is visible in the Physical Viewer before making a snapshot. Note: This option eliminates the need to first turn on the display of the congestion map.
file	Specifies the name of the file in which to save the snapshot.
-no_fit	Specifies to not fit the snapshot to the full screen.
-overwrite	Specifies to overwrite an existing file. If you omit this option, you'll get a message that indicates that the file exists.
-pin_density	Overlays the pin density map on top of what is visible in the Physical Viewer before making a snapshot.
-ps	Specifies to create the file in PostScript format. If this option is omitted, a GIF file will be created.
-utilization	Overlays a utilization map on the part of the design that is visible in the Physical Viewer before making a snapshot. Note: This option eliminates the need to first turn on the display of the utilization map.

Command Reference for Encounter RTL Compiler GUI Text

Examples

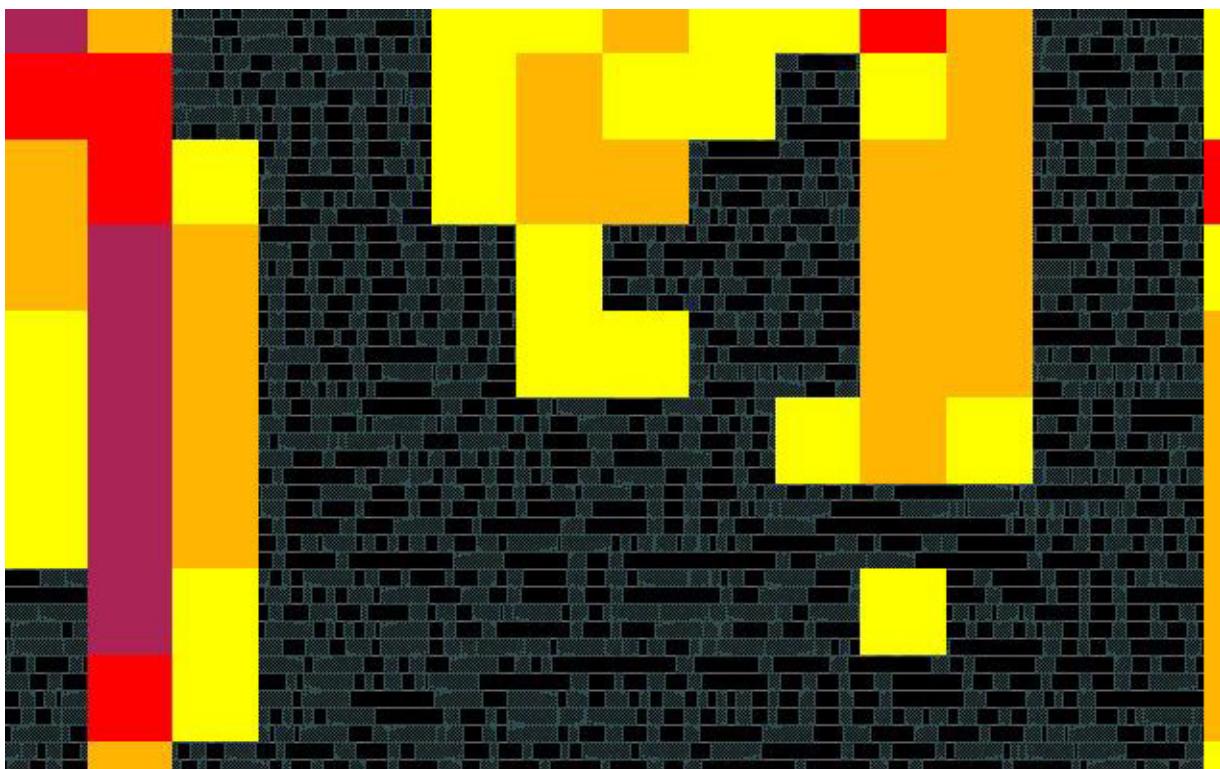
- The following command saves a snapshot of the utilization map in the util.gif file.

```
gui_pv_snapshot -utilization util.gif
```



- The following command saves a snapshot of the congestion map in the file with basename congest. Since the -ps option is not specified, the file is specified in GIF format.

```
gui_pv_snapshot -congestion congest
```



gui_pv_steiner_tree

```
gui_pv_steiner_tree  
  [-append] {instance | port}...
```

Shows the steiner tree from the output pins of the specified instance or from the specified port.

Options and Arguments

-append	Adds the new steiner trees to the existing ones.
{instance port}	Specifies the name of the instance or port from where to draw the steiner trees.

gui_pv_update

```
gui_pv_update [string] [-design design]
```

Specifies the objects to be updated in the Physical Viewer.

Options and Arguments

-design <i>design</i>	Specifies the design for which to update the GUI.
<i>string</i>	Specifies the objects to be updated in the Physical Viewer. You can specify row or region. By default, all object will be updated.

Example

In the following example, the first two command create a new row without GUI update. The third command request a GUI update to show the new rows.

```
create_row -no_update ...  
create_row -no_update ...  
gui_pv_update row
```

gui_pv_zoom_box

`gui_pv_zoom_box llx lly urx ury`

Zooms to the specified box in the Physical Viewer.

Options and Arguments

`llx lly urx ury` Specifies the lower left and upper right coordinates of the box to zoom in to. You can specify floating numbers.

The coordinates are specified in user units.

Command Reference for Encounter RTL Compiler GUI Text

gui_pv_zoom_fit

`gui_pv_zoom_fit`

Performs a “zoom fit” command in the Physical Viewer.

gui_pv_zoom_in

`gui_pv_zoom_in`

Performs a “zoom in” command in the Physical Viewer.

gui_pv_zoom_out

`gui_pv_zoom_out`

Performs a “zoom out” command in the Physical Viewer.

gui_pv_zoom_to

`gui_pv_zoom_to`

Zooms to the bounding box around selected objects in the Physical Viewer.

Command Reference for Encounter RTL Compiler

GUI Text

Chipware Developer

- [cwd on page 160](#)
- [cwd check on page 161](#)
- [cwd create check on page 165](#)
- [cwd report check on page 167](#)
- [hdl_create on page 169](#)
- [hdl_create binding on page 170](#)
- [hdl_create component on page 172](#)
- [hdl_create implementation on page 174](#)
- [hdl_create library on page 176](#)
- [hdl_create operator on page 177](#)
- [hdl_create package on page 178](#)
- [hdl_create parameter on page 180](#)
- [hdl_create pin on page 182](#)

cwd

```
cwd {check | create_check | report_check}
```

Controls the ChipWare Developer (CWD) Linter in the ChipWare developer framework.

Options and Arguments

check	Invokes the CWD Linter.
create_check	Registers a check to the Linter.
report_check	Reports information about various check names and check points.

Related Information

Related commands:	cwd check on page 161
	cwd create_check on page 165
	cwd report_check on page 167

cwd check

```
cwd check [-effort string]
[-skip
 [ hdl_lib | hdl_comp | hdl_pack | hdl_oper
 | hdl_arch | hdl_impl | hdl_bind | hdl_param
 | hdl_pin]... ]
[ [-summary [-verbose] | -quiet ]
 [ hdl_lib | hdl_comp | hdl_pack | hdl_oper
 | hdl_arch | hdl_impl | hdl_bind | hdl_param
 | hdl_pin]... ]
[> file]
```

Exercises checking rules and summarizes the outcome in various degrees of verboseness.
The cwd check command can run on one or more of any of the following hdl_objects:

- *hdl_lib*
- *hdl_oper*
- *hdl_comp*
- *hdl_bind*
- *hdl_impl*
- *hdl_param*
- *hdl_pin*
- *hdl_pack*
- *hdl_arch*

The RTL Compiler path to the hdl_objects to be checked can either be an absolute path:

```
rc:/> cwd check /hdl_libraries/CW/components/CW_add
```

Or it can be a relative path with respect to the current working directory:

```
rc:/> cd /hdl_libraries/CW/components
rc:/> cwd check CW_add
```

You can use wild cards for specifying multiple hdl_objects:

```
rc:/> cwd check /hdl_libraries/CW/components/*add*
```

or:

```
rc:/> cd /hdl_libraries/CW/components
rc:/> cwd check *add*
```

Command Reference for Encounter RTL Compiler

Chipware Developer

You can specify multiple directories at the same time:

```
rc:/> cwd check {/hdl_libraries/CW /hdl_libraries/DW02}
```

By default, `cwd check` checks all `hdl_objects` underneath the specified set of `hdl_objects`. That is, it traverses the directory tree hierarchically and exercises all checks of all `hdl_objects` it traverses.

Options and Arguments

<code>-effort string</code>	Specifies the effort level. There are two effort levels: <code>low</code> and <code>medium</code> . The default effort level is <code>low</code> .
	<code>Low</code> — CWD linter runs checking rules that are registered as <code>low</code> effort. That is, it runs those checking rules that do not require reading any HDL code (of synthesis models).
	<code>Medium</code> — CWD linter runs checking rules that are registered as <code>low</code> and <code>medium</code> effort. That is, it runs those checking rules that may require parsing HDL code (of synthesis models) but do not require elaborating it.
	With each <code>medium</code> effort level check, the CWD linter automatically loads the HDL code before performing the check. For example, a check at this effort level may look at the <code>hdl_arch</code> of an <code>hdl_impl</code> and examine ordering of pins and parameters.
<code>file</code>	Specifies the filename to store the output of the command.
<code>hdl_lib hdl_comp hdl_pack hdl_oper hdl_arch hdl_impl hdl_bind hdl_param hdl_pin</code>	Specifies the <code>hdl_objects</code> to check.
<code>-quiet</code>	Only reports error and warning messages, if any. This is the recommended verbosity level when CWD linting is part of a routine process without any error expectations.
<code>-skip {hdl_lib hdl_comp hdl_pack hdl_oper hdl_arch hdl_impl hdl_bind hdl_param hdl_pin}</code>	Specify one or more <code>hdl_objects</code> to skip.

Command Reference for Encounter RTL Compiler

Chipware Developer

-summary	First reports error or warning messages, if any, and then produces a summary table of the pass/fail count of each checking rules exercised. This level of detail is the default verbosity level.
-verbose	Produces a detailed report. In addition to the information produced by the -summary option, it also reports the pass/fail status of each check process exercised on each hdl_object.

Examples

- The following example runs checking rules on the CW libraries as well as all the other hdl_objects under it. The CWD linter will run all default (low-effort) mode checking rules up to the severity specified by the information_level attribute. A summary will be produced at the end.

```
rc:/> get_attribute information_level
1
rc:/> cwd check /hdl_libraries/CW
Check_name          Passed Failed
-----
component_location      146    0
component_sim_model_location 128    0
component_syn_model_is_vhdl 146    0
implementation_legality_formula 147    0
implementation_location     147    0
implementation_prelab_script_location 147    0
non_builtin_implementation_location 147    0
package_default_location     1    0
package_default_location_filesize 1    0
parameter_formula          472    0
pin_bit_width              1136   0
pin_parameter_in_bit_width 1114   0
-----
```

Command Reference for Encounter RTL Compiler

Chipware Developer

- The following example runs checking rules on all the hdl_objects under parameters and produces a verbose report:

```
rc:/> cwd check /hdl_libraries/CW/components/CW_mult/parameters/* -verbose
      checking param wA
      Check ::cwd::parameter_formula::check_proc passed on /hdl_libraries/CW/
      components/CW_mult/parameters/wA
      checking param wB
      Check ::cwd::parameter_formula::check_proc passed on /hdl_libraries/CW/
      components/CW_mult/parameters/wB
      Check_name      Passed Failed
      -----
      parameter_formula    2        0
      -----
```

- The following example will check the CW_add component, but will skip checking its bindings and implementations:

```
rc:/> cd /hdl_libraries/CW/components/CW_add
rc:/> cwd check . -skip { /hdl_libraries/CW/components/CW_add/bindings/* \
      /hdl_libraries/CW/components/CW_add/implementations/* }
```

Related Information

[Checking Rules in ChipWare Developer](#)

cwd create_check

```
cwd create_check
  -check_name string
  -severity integer
  -description string
  -checklist string
  [-effort {low|high}] [-force] [> file]
```

Registers user-defined checking rules for the CWD linter.

Options and Arguments

- check_name *string* Specifies an unique name for a new checking rule.
- checklist *string* Specifies, as a Tcl list of Tcl lists, checkpoint and Tcl proc pairs. The specification therefore would be in the form:
 - checklist {{point_1 proc_1} {point_2 proc_2}}Every sub-list has two elements. The first element is the name of a check point, defined by RTL Compiler. The second element is the name of a Tcl proc, defined by the user.
Each sub-list specifies a check proc that is to be called at a certain check point. This check proc will be executed every time the flow reaches this check point.
This Tcl list has one or more sub-lists. One checking rule can be associated with one or more check points.
The check procs may or may not be allowed to parse or elaborate the HDL code of the synthesis model, depending on the effort level of this checking rule. The check procs may print out error, warning, or info messages. Each check proc should return a string whose value is either PASS or FAIL.
- description *string* Specifies a character string that concisely describes what this checking rule examines.
- effort {low|high} Specifies the effort level.
Default: low
low — The check is not allowed to parse the HDL code of the synthesis models.

It can check correctness, consistency, or completeness of the CWD registration, including availability of UNIX files referred to by the `location` and `sim_model` attributes.

`medium` — The check can read, load, and parse the HDL code of the synthesis models, but it cannot elaborate or synthesize the HDL code. When exercising such a checking rule, the HDL code is automatically loaded before checking is performed.

<code>file</code>	Specifies the filename to store the output of the command.
<code>-force</code>	Removes the existing checkname and adds the current specification. Alternatively, the same checkname will now correspond to the new definition.
<code>-severity integer</code>	<p>Specifies the severity level of the message produced by this checking rule. The possible values (0 through 10) are the same as those for the <code>information_level</code> attribute.</p> <ul style="list-style-type: none"><code>-severity 0</code>: It is an error message to violate this rule<code>-severity 1</code>: It is a warning message to violate this rule<code>-severity 2</code>: It is a level 2 info message to violate this rule <p>Severity levels 3 through 10 are all info messages at their respective levels.</p>

Example

The following example defines the name of the check to be `arch_pin_order`. It is a medium effort level check: it has to be performed after the HDL code has been loaded. The severity of the check is 0, which means it is an error if this check fails. This checking rule is associated with two checkpoints, `HDL_ARCH_PINS_SCANNED` and `HDL_COMP_PINS_SCANNED`. At the `HDL_ARCH_PINS_SCANNED` checkpoint, a Tcl proc named `check_arch_pin_order` is to be called to perform this check. At the `HDL_COMP_PINS_SCANNED` checkpoint, a Tcl proc named `check_component_pin_order` is to be called to perform this check.

```
rc> cwd create_check -check_name "arch_pin_order" -effort "medium" \
    -severity 0 -description "Check Whether the order of pins specified \
    in the synthesis model is consistent with what is defined in the \
    registration script"
    -checklist { {HDL_ARCH_PINS_SCANNED check_arch_pin_order} \
    {HDL_COMP_PINS_SCANNED check_component_pin_order} }
```

Related Information

[Checking Rules in ChipWare Developer](#)

cwd report_check

```
cwd report_check  
  [-checkpoint string] [-checkname string]  
  [-max_width string] [> file]
```

Reports the registered checking rules. With each checking rule, it lists the:

- Name of the checking rule
- Checkpoint(s) the rule is associated with
- Check proc(s) attached to the associated checkpoint(s)
- Effort level of the rule
- Severity level of the rule
- Description string of the rule

Note: The `-checkname` and `-checkpoint` options cannot be both used simultaneously.

Options and Arguments

<code>-checkname string</code>	Specifies, by name, a set of checking rules to report.
<code>-checkpoint string</code>	This switch specifies a set of checkpoints to report.
<code>-max_width string</code>	Limits the width of the columns in the table produced by this command. Limiting the width of a column to zero means removing that column from the table. This option takes a Tcl list of Tcl lists. Each sub-list represents a column in the table produced by this command. Each sub-list should have two elements: the first being name of the column (as seen in the report) and the second being an integer representing the maximum number of characters allowed for this column in the table.
<code>file</code>	Specifies the filename to store the output of the command.

Command Reference for Encounter RTL Compiler

Chipware Developer

Examples

- The following example reports details about the checking rule `arch_pin_order`, which uses two check procs to associate with two checkpoints.

```
rc:/> cwd report_check -checkname {arch_pin_order} -max_width \
    {{Check_name 14} {Checkpoint 12} {Check_proc 15} {Effort 3} \
    {Severity 4} {Description 20}}
```

This example reports details about the checking rule `arch_pin_order`, which uses two check procs to associate with two checkpoints.

Check_name	Checkpoint	Check_proc	Effort	Severity	Description
arch_pin_order	HDL_ARCH_PIN S_SCANNED	::cwd::arch_pin order::cwd_proc ium c	0	med	Check whether the order of pins specified in the synthesis model is consistent with what is defined in the registration script
	HDL_COMP_PIN S_SCANNED	::cwd::componen t_pin_order::ch eck_proc			

- The following example reports details about the checkpoint `HDL_OPER_BINDINGS_SCANNED`:

```
rc:/> cwd report_check -checkpoint {HDL_OPER_BINDINGS_SCANNED}
    -max_width {{Check_name 10} {Checkpoint 15} {Check_proc 15} \
    {Effort 3} {Severity 4} {Description 20}}
```

Check_name	Checkpoint	Check_proc	Effort	Severity	Description
operator_b	HDL_OPER_BINDIN indings GS_SCANNED	::cwd::operator bindings::che k_proc	1	low	check that for every hdl_bindings defined for the hdl_operator there is at least one attribute is set to false

- The following example reports all checking rules that have been registered:

```
rc:/> cwd report_check -checkname {*} 
```

- This reports info about all checkpoints:

```
rc:/> cwd report_check -checkpoint {*} 
```

- The following example reports all checking rules whose checkname contains string "pin":

```
rc:/> cwd report_check -checkname {*pin*}
```

- The following example reports information about the `arch_pin_order` and `arch_parameter_order` checking rules:

```
rc:/> cwd report_check -checkname {arch_pin_order arch_parameter_order}
```

hdl_create

```
hdl_create { binding | component | implementation | library  
| operator | package | parameter | pin}
```

Creates an HDL object for ChipWare developer.

Options and Arguments

binding	Creates a binding between a synthetic operator and a ChipWare component.
component	Creates a ChipWare component.
implementation	Creates a synthesis model for a ChipWare component.
library	Creates a synthetic library to hold ChipWare components, bindings, and implementations.
operator	Creates a synthetic operator.
package	Creates a package in the Design Information Hierarchy to hold the contents of a VHDL package.
parameter	Creates a parameter for a synthetic ChipWare component
pin	Creates an input/output/inout pin for a synthetic operator or component

Related Information

Related commands:	hdl_create binding on page 170 hdl_create component on page 172 hdl_create implementation on page 174 hdl_create library on page 176 hdl_create operator on page 177 hdl_create package on page 178 hdl_create parameter on page 180 hdl_create pin on page 182
-------------------	--

hdl_create binding

```
hdl_create binding binding_name
    -operator operator_name
    [hdl_comp | bindings]
```

Creates a binding between a synthetic operator and a synthetic module. A synthetic module is also known as a ChipWare component.



Tip
You can save run-time if you `cd` to the *component name* directory and issue the command instead of specifying the entire component path name.

Options and Arguments

<i>binding_name</i>	Specifies the name of the binding that will be created.
<i>hdl_comp</i> <i>bindings</i>	Specifies the path name of the component that holds this binding.
-operator	Specifies the synthetic operator that will be bound by this binding.

Examples

- The following examples both create a binding named `test1` for the `MY_MULT_OP` operator. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command.

```
rc:/hdl_libraries/my_CW/components/my_CW_mult> hdl_create binding \
    test1 -operator MY_MULT_OP
rc:/hdl_libraries/my_CW/components/my_CW_mult/bindings> ls
./      test1
```

- This example also creates a binding named `test1`. However, the command is issued from the root directory and therefore assumes a run-time penalty.

```
rc:/> hdl_create binding test1 -operator MY_MULT_OP \
    /hdl_libraries/my_CW/components/my_CW_mult
rc:/> ls /hdl_libraries/my_CW/components/my_CW_mult/bindings
./      test1
```

Related Information

CWD Component in *ChipWare Developer*

Related commands:

[hdl_create_component](#) on page 172
[hdl_create_implementation](#) on page 174
[hdl_create_library](#) on page 176
[hdl_create_operator](#) on page 177
[hdl_create_package](#) on page 178
[hdl_create_parameter](#) on page 180
[hdl_create_pin](#) on page 182

hdl_create component

```
hdl_create component component_name
    [hdl_lib | components]
```

Creates a ChipWare component.

Options and Arguments

component_name Specifies the name of the component that will be created.

hdl_lib | *components*

Specifies the path name of the library that holds this component.

Examples

- The following examples both create a component named `CW_sweet_div`. However, the first example will save run-time over the second by cd'ing into the `components` directory and issuing the command:

```
rc:/hdl_libraries/CW/components> hdl_create component CW_sweet_div
rc:/hdl_libraries/CW/components> ls
...
CW_sweet_div
...
```

- This example also creates a component named `CW_sweet_div`. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create component CW_sweet_div /hdl_libraries/CW/
rc:/> ls /hdl_libraries/CW/components
...
CW_sweet_div
...
```

Related Information

See the following sections in *ChipWare Developer*

- [CWD Component](#)
- [ChipWare Registration](#)

Related commands:

- [hdl_create_binding](#) on page 170
- [hdl_create_implementation](#) on page 174
- [hdl_create_library](#) on page 176
- [hdl_create_operator](#) on page 177
- [hdl_create_package](#) on page 178
- [hdl_create_parameter](#) on page 180
- [hdl_create_pin](#) on page 182

hdl_create implementation

```
hdl_create implementation implementation_name
    [-v1995 | -v2001 | -vhdl87 | -vhdl93]
    [hdl_comp | implementations]
```

Creates an implementation for a ChipWare component. All implementations created with this command have a default priority of 1. A ChipWare implementation is also known as an architecture of the component. You must specify a language version.

Options and Arguments

implementation_name

Specifies the name of the implementation that will be created.

hdl_comp | *implementation*

Specifies the path name of the component that owns this implementation.

[-v1995 | -v2001 | -vhdl87 | -vhdl93]

Specifies the language version for the RTL code.

Example

- Both of the following examples create the `krystal` implementation in VHDL 1993 format for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

```
rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create implementation \
    krystal -vhdl93
rc:/hdl_libraries/CW/components/CW_sweet_div/implementations> ls
./      krystal
```

- This example also creates an implementation named `krystal` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create implementation krystal -vhdl93 \
    /hdl_libraries/CW/components/CW_sweet_div
rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/implementations/
./      krystal
```

Related Information

See the following sections in *ChipWare Developer*

- [CWD Component](#)
- [Chipware Installation and Registration](#)

Affects this attribute: [priority](#)

Related commands:

- [hdl_create_binding](#) on page 170
- [hdl_create_component](#) on page 172
- [hdl_create_library](#) on page 176
- [hdl_create_operator](#) on page 177
- [hdl_create_package](#) on page 178
- [hdl_create_parameter](#) on page 180
- [hdl_create_pin](#) on page 182

hdl_create library

`hdl_create library library_name`

Creates an HDL library. An HDL library can be a library of ChipWare components, a library of synthetic operators, or a VHDL library.

Options and Arguments

library_name Specifies the name of the library that will be created.

Related Information

See the following sections in *ChipWare Developer*

- [CWD Component](#)
- [ChipWare Installation and Registration](#)

Related commands:

<u>hdl_create binding</u> on page 170
<u>hdl_create component</u> on page 172
<u>hdl_create implementation</u> on page 174
<u>hdl_create operator</u> on page 177
<u>hdl_create package</u> on page 178
<u>hdl_create parameter</u> on page 180
<u>hdl_create pin</u> on page 182

hdl_create operator

```
hdl_create operator operator_name
    [-signed | -unsigned]
```

Creates a synthetic operator. The default operator type is unsigned.

Options and Arguments

<i>operator_name</i>	Specifies the name of the synthetic operator that will be created.
-signed	Specifies the created operator to be a signed operator.
-unsigned	Specifies the created operator to be an unsigned operator. This is the default setting.

Related Information

[Synthetic Operator in ChipWare Developer](#)

Related commands:	hdl_create binding on page 170
	hdl_create component on page 172
	hdl_create implementation on page 174
	hdl_create library on page 176
	hdl_create package on page 178
	hdl_create parameter on page 180
	hdl_create pin on page 182

hdl_create package

```
hdl_create package pkg_name
    -path path_to_pkg
    [hdl_lib | packages]
```

Registers a VHDL package in the ChipWare Developer framework. Packages that are not registered are deleted after elaboration. However, registered packages are never deleted and their information can be further considered during synthesis as opposed to just during elaboration.

Registered packages are in the same location within RTL Compiler as non-registered packages:

```
/hdl_libraries/library_name/packages/
```

Options and Arguments

hdl_lib | *packages*

Specifies the path name of the library that holds this package.

-path *path_to_pkg* Specifies the UNIX path name of the package to register.

pkg_name Specifies the name of the package that will be created.

Examples

- Both of the following examples create the `numeric_std` package for the `ieee` library. However, the first example will save run-time over the second by cd'ing into the library name directory and issuing the command:

```
rc:/hdl_libraries/ieee/packages> hdl_create package numeric_std -path \
    /home/krystal/vhdl/packages/numeric_std.vhd
rc:/hdl_libraries/ieee/packages> ls
./      numeric_std
```

- This example also creates a package named `numeric_std` for the same library. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create package numeric_std -path /home/krystal/vhdl/packages \
    /hdl_libraries/ieee/packages/numeric_std
rc:/> ls /hdl_libraries/ieee/packages/
./      numeric_std
```

Related Information

Related commands:

[hdl_create binding](#) on page 170

[hdl_create component](#) on page 172

[hdl_create implementation](#) on page 174

[hdl_create library](#) on page 176

[hdl_create operator](#) on page 177

[hdl_create parameter](#) on page 180

[hdl_create pin](#) on page 182

hdl_create parameter

```
hdl_create parameter parameter_name  
[-hdl_invisible]  
[hdl_comp | parameters]
```

Creates a parameter for a ChipWare component. The created parameter will be a `hdl_param` object type and located under `../component_name/parameters`. The default `hdl_parameter` attribute value for parameters created with this command will be `true`. However, if the `-hdl_invisible` option is specified, the default value becomes `false`.

Options and Arguments

hdl_comp | parameters

Specifies the path name of the component that holds this parameter.

-hdl_invisible Specifies that the parameter cannot be accessed from the HDL. The value of the `hdl_parameter` attribute for this parameter becomes `false` with this option.

parameter_name Specifies the name of the parameter that will be created.

Examples

- Both of the following examples create the `WIDTH` parameter for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

```
rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create parameter WIDTH  
rc:/hdl_libraries/CW/components/CW_sweet_div/parameter> ls  
. / WIDTH
```

- This example also creates a parameter named `WIDTH` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create parameter WIDTH /hdl_libraries/CW/components/CW_sweet_div  
rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/parameters/  
. / WIDTH
```

Related Information

See the following sections in *ChipWare Developer*

- [CWD Component](#)
- [ChipWare Installation and Registration](#)

Affects this attribute: [hdl_parameter](#)

Related commands:

- [hdl_create_binding](#) on page 170
- [hdl_create_component](#) on page 172
- [hdl_create_implementation](#) on page 174
- [hdl_create_library](#) on page 176
- [hdl_create_operator](#) on page 177
- [hdl_create_package](#) on page 178
- [hdl_create_pin](#) on page 182

hdl_create pin

```
hdl_create pin pin_name
  {-input | -output | -inout}
  [pins | hdl_oper | hdl_comp]
```

Creates a pin for either a ChipWare component or a synthetic operator. You must specify a pin direction.

Options and Arguments

<code>-inout</code>	Specifies that the created pin will be an bidirectional pin.
<code>-input</code>	Specifies that the created pin will be an input pin.
<code>-output</code>	Specifies that the created pin will be an output pin.
<code><i>pin_name</i></code>	Specifies the name of the pin that will be created.
<code><i>pins</i> <i>hdl_oper</i> <i>hdl_comp</i></code>	Specifies the path name of the component or synthetic operator that holds the created pin.

Examples

- Both of the following examples create the `div_in` input pin for the `CW_sweet_div` component. However, the first example will save run-time over the second by `cd`'ing into the component name directory and issuing the command:

```
rc:/hdl_libraries/CW/components/CW_sweet_div> hdl_create pin -input div_in
rc:/hdl_libraries/CW/components/CW_sweet_div/pins/> ls
./      div_in
```

- This example also creates an input pin named `div_in` for the same component. However, the command is issued from the root directory and therefore assumes a run-time penalty:

```
rc:/> hdl_create pin -input div_in
rc:/> ls /hdl_libraries/CW/components/CW_sweet_div/pins/
./      div_in
```

Related Information

See the following sections in *ChipWare Developer*

- [CWD Component](#)
- [ChipWare Installation and Registration](#)
- [Synthetic Operator](#)

Related commands:

- [hdl_create binding](#) on page 170
- [hdl_create component](#) on page 172
- [hdl_create implementation](#) on page 174
- [hdl_create library](#) on page 176
- [hdl_create operator](#) on page 177
- [hdl_create package](#) on page 178
- [hdl_create parameter](#) on page 180

Command Reference for Encounter RTL Compiler
Chipware Developer

Input and Output

- [decrypt](#) on page 188
- [encrypt](#) on page 189
- [export_critical_endpoints](#) on page 192
- [read_cpf](#) on page 194
- [read_db](#) on page 195
- [read_def](#) on page 196
- [read_dfm](#) on page 197
- [read_dft_abstract_model](#) on page 199
- [read_encounter](#) on page 200
- [read_hdl](#) on page 201
- [read_io_speclist](#) on page 205
- [read_netlist](#) on page 206
- [read_saif](#) on page 208
- [read_sdc](#) on page 209
- [read_spf](#) on page 211
- [read_tcf](#) on page 212
- [read_vcd](#) on page 213
- [restore_design](#) on page 214
- [split_db](#) on page 216
- [write_atpg](#) on page 217
- [write_bsdl](#) on page 218
- [write_compression_macro](#) on page 219

Command Reference for Encounter RTL Compiler

Input and Output

- [write_cpf](#) on page 220
- [write_db](#) on page 221
- [write_def](#) on page 223
- [write_design](#) on page 224
- [write_dft_abstract_model](#) on page 226
- [write_do_ccd](#) on page 227
- [write_do_ccd_compare_sdc](#) on page 228
- [write_do_ccd_generate](#) on page 230
- [write_do_ccd_validate](#) on page 234
- [write_do_clp](#) on page 235
- [write_do_lec](#) on page 238
- [write_do_verify_cdc](#) on page 241
- [write_encounter](#) on page 243
- [write_et_atpg](#) on page 246
- [write_et_bsv](#) on page 247
- [write_et_mbist](#) on page 250
- [write_et_rrfa](#) on page 251
- [write_ets](#) on page 252
- [write_ett](#) on page 253
- [write_forward_saif](#) on page 254
- [write_hdl](#) on page 255
- [write_io_speclist](#) on page 259
- [write_saif](#) on page 260
- [write_scandef](#) on page 261
- [write_script](#) on page 262
- [write_sdc](#) on page 265
- [write_sdf](#) on page 269
- [write_set_load](#) on page 274

Command Reference for Encounter RTL Compiler

Input and Output

- [write_spf](#) on page 275
- [write_sv_wrapper](#) on page 276
- [write_tcf](#) on page 279
- [write_template](#) on page 280

decrypt

```
decrypt [-keydb path] file
```

Decrypts and evaluates a Tcl file that was encrypted with the [encrypt](#) command.

Options and Arguments

<i>file</i>	Specifies the name of the Tcl file to decrypted and evaluated.
<i>-keydb path</i>	Sets the NCProtect_KEYDB environment variable to the directory containing the public key needed to decrypt the file. If you omit this option, you need to set the NCProtect_KEYDB environment variable before you run the <code>decrypt</code> command.

Example

The first command encrypts the Tcl file `my_script.g`. The second command decrypts the `my_script_encr.g` file. These commands are normally executed in different RC sessions.

```
rc:/> encrypt -tcl my_script.g > my_script_encr.g  
rc:/> decrypt my_script_encr.g
```

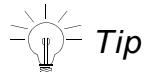
Related Information

Related command: [encrypt](#) on page 189

encrypt

```
encrypt inputfile_name
    [-vlog | -vhdl | -tcl] [-pragma]
    [> file]
```

Uses the NC Protect protection scheme to encrypt the specified HDL or Tcl files.



To load encrypted HDL files, use the `read_hdl` command.

The command to source an encrypted Tcl file depends on the file extension of the encrypted Tcl file:

- ❑ If the encrypted file does not have the `.etf` extension, use the `decrypt` command.
- ❑ If the encrypted file has the `.etf` extension, you can use the `source` command.

Options and Arguments

<i>input_file_name</i>	Specifies the file to be encrypted.
<i>file</i>	Specifies the name of the encrypted file. By default, the encrypted file is printed to standard out.
-pragma	Specifies to only encrypt the text between the <code>protect begin</code> and <code>protect end</code> NC Protect pragmas.
-tcl	Specifies that the file to be encrypted is a Tcl file. To hide the body of the Tcl scripts, make sure to code procedures using <code>hidden_proc</code> .
-vhdl	Uses VHDL style comments for NC Protect pragmas.
-vlog	Uses Verilog style comments for NC Protect pragmas. This is the default option.

Example

- The following example encrypts the `ksable.vhdl` VHDL file, with VHDL constructs, to a file named `ksable_encrypted.vhdl`. The encrypted file is then loaded.

```
rc:/> encrypt -vhdl ksable.vhdl > ksable_encrypted.vhdl
rc:/> read_hdl -vhdl ksable_encrypted.vhdl
```

Command Reference for Encounter RTL Compiler

Input and Output

- The following example illustrates Verilog code with Verilog style NC Protect pragmas. You must specify //pragma protect before specifying the beginning (//pragma protect begin) and ending (//pragma protect end) pragmas.

```
module secret_func (y, a, b);
    parameter w = 4;
    input [w-1:0] a, b;
    output [w-1:0] y;
// pragma protect
// pragma protect begin
    assign y = a & b;
// pragma protect end
endmodule
```

Specifying the -vlog and -pragma options together will only encrypt the text between the pragmas. The following command encrypts the original verilog file (*ori.v*) that contained the NC Protect pragmas. The encrypted file is called *enc.v*.

```
rc:> encrypt -vlog -pragma org.v > enc.v
```

- The following example illustrates VHDL code with VHDL style NC Protect pragmas. You must specify --pragma protect before specifying the beginning (--pragma protect begin) and ending (--pragma protect end) pragmas.

```
entity secret_func is
    generic (w : integer := 4);
    port ( y: out bit_vector (w-1 downto 0);
           a, b: in bit_vector (w-1 downto 0) );
end;

-- pragma protect
-- pragma protect begin
architecture rtl of secret_func is
begin
    y <= a and b;
end;
-- pragma protect end
```

Specifying the -vhdl and -pragma options together will only encrypt the text between the pragmas. The following command encrypts the original VHDL file (*ori.vhd1*) that contained the NC Protect pragmas. The encrypted file is called *enc.vhd1*:

```
rc:/> encrypt -vhdl -pragma org.vhd1 > enc.vhd1
```

- The following example shows a Tcl script (*test.tcl*) with two procedures: the first procedure starting with *proc*, the second one starting with *hidden_proc*. When the script is encrypted, no info will be returned for the *im_hidden* procedure.

```
# pragma protect
# pragma protect begin
proc im_visible {args} {
    # 'info' command will return data for this proc
}
hidden_proc im_hidden {args} {
    # 'info' command will not return data for this proc
}
# pragma protect end
```

Command Reference for Encounter RTL Compiler

Input and Output

```
rc:/> encrypt -tcl test.tcl > test_enc.tcl
rc:/> info body im_hidden
rc:/> info body im_visible
# 'info' command will return data for this proc
```

Related Information

Related command: [decrypt](#) on page 188

Command Reference for Encounter RTL Compiler

Input and Output

export_critical_endpoints

```
export_critical_endpoints
  -rc_file string -fe_file string
  [-group | -no_group] [-verbose]
  [-percentage_of_endpoints integer]
  [-no_of_bins integer]
  [-percentage_difference integer] [-rtl]
  [-design string] [> file]
```

Generates a path adjust file, which allows synthesis to provide better timing closure results to Encounter.

Options and Arguments

<code>-design string</code>	Specifies the module name.
<code>-fe_file string</code>	Specifies the First Encounter (FE) slack report that you want to compare.
<code>file</code>	Specifies the name of the file to write the report.
<code>[-group -nogroup]</code>	Specifies whether to group endpoints into bins for path_adjust or not. <i>Default:</i> -group
<code>-no_of_bins integer</code>	Specifies the number of bins to group the endpoints for compression. <i>Default:</i> 10 bins each for tighten and relax
<code>-percentage_difference integer</code>	Specifies the percentage difference between the endpoints to be path adjusted (with the path_adjust command). <i>Default:</i> 70%
<code>-percentage_of_endpoints integer</code>	Specifies the percentage of endpoints to be constrained or relaxed. <i>Default:</i> 20%
<code>-rc_file string</code>	Specifies the RTL Compiler endpoint report that you want to compare.

Command Reference for Encounter RTL Compiler

Input and Output

-rtl Writes a path adjust file that can be applied to the RTL.
-verbose Specifies a verbose report.

Related Information

Path Adjust Flows in Encounter RTL Compiler Synthesis Flows

read_cpf

Refer to [read_cpf](#) in [Chapter 12, “Advanced Low Power Synthesis.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

read_db

```
read_db  
  {db_file | -from_tcl string}  
  [-quiet] [-verbose]
```

Loads the specified database file or Tcl object.

If the database contains setup information, the setup is restored as well. If the setup was written to a separate script, you must source that script before you read the database file.

Options and Arguments

<i>db_file</i>	Specifies the name of the database file to be read.
<i>-from_tcl string</i>	Specifies to read the specified Tcl object.
<i>-quiet</i>	Suppresses any warning messages. Error messages are printed.
<i>-verbose</i>	Enables verbose output while reading in the database file.

Related Information

[Using the RTL Compiler Database](#) in *Using Encounter RTL Compiler*

Related commands:

split_db on page 216
write_db on page 221

Command Reference for Encounter RTL Compiler

Input and Output

read_def

Refer to [read_def](#) in [Chapter 9, “Physical.”](#)

read_dfm

```
read_dfm coefficients_filename
```

Loads the coefficients file. You can only load one file at a time. After the coefficients file is loaded, RTL Compiler will annotate the defect probability of any matching cells between the coefficients file and the timing library.

Options and Arguments

coefficients_filename

Specifies the name of the coefficients file.

Example

- A DFM file is described in XML format. The following example shows what a DFM file might look like:

```
<?xml version="1.0"?>

<yield_file>
  <title> file with probabilities of failure of each library cell </title>
  <cell_probability>
    <cell> inv1
      <instance> 0.000000026309750 </instance>
      <systematic> 0.0000000000000000 </systematic>
    </cell>

    <cell> fflop
      <instance> 0.000000153055338 </instance>
      <systematic> 0.0000000000000000 </systematic>
    </cell>
    <cell> nand2
      <instance> 0.000000044800000 </instance>
      <systematic> 0.0000000000000000 </systematic>
    </cell>
  </cell_probability>
</yield_file>
```

- The following example loads two coefficient files:

```
rc:/> read_dfm test1.dfm
rc:/> read_dfm test2.dfm
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Design For Manufacturing Flow in *Encounter RTL Compiler Synthesis Flows*](#)

Affects these commands: [report gates -yield](#)

[report yield](#)

Affects this attribute: [yield](#)

Related attribute: [optimize_yield](#)

Command Reference for Encounter RTL Compiler

Input and Output

read_dft_abstract_model

Refer to [read_dft_abstract_model](#) in [Chapter 10, “Design for Test.”](#)

read_encounter

Refer to [read_encounter](#) in [Chapter 9, “Physical.”](#)

read_hdl

```
read_hdl file_list
  [ {-v1995 | -v2001 | -sv | -vhdl }
   [-library library_name[=library_name2]...]
   | -netlist]
   [-define macro=value]... file_list
```

Loads one or more HDL files in the order given into memory. Files containing macro definitions should be loaded before the macros are used. Otherwise, there are no ordering of constraints.

If you do not specify either the `-v1995`, `-v2001`, `-sv` or the `-vhdl` option, the default language format is that specified by the `hdl_language` attribute. The *default* value for the `hdl_language` attribute is `-v1995`.

The HDL files can contain structural code for combining lower level modules, behavioral design specifications, or RTL implementations.

You can automatically read in or write out a compressed HDL file in gzip format. For example:

```
read_hdl sample.v.gz
write_hdl -g sample.v.gz
```

When you load a parameterized Verilog module or VHDL architecture, each parameter in the module or architecture will be identified as an `hdl_param` object and located under `./architecture_name/parameters`. The default `hdl_parameter` attribute value for these parameters will be `true`.

Use the `rc -E -f <your script>` command to specify that RTL Compiler automatically quit if a script error is detected when reading in HDL files instead of holding at the `rc>` prompt.

Options and Arguments

`-define macro=value` Defines a Verilog macro with the specified *value*, which is equivalent to the `'define macro value`.

Note: You can also define a macro definition list.

`file_list` Specifies the name of the HDL files to load. If several files must be loaded, specify them in a string.

Note: The files can be encrypted.

The host directory where the HDL files are looked for is specified via the `hdl_search_path` root attribute.

Command Reference for Encounter RTL Compiler

Input and Output

`-library library_name[=library_name2] . . .`

Specifies the name of the Verilog or VHDL library in which the definitions will be stored.

A virtual directory with this name will be created in the `hdl_libraries` directory of the design hierarchy if it does not already exist.

If you specify multiple libraries, they become multiple names (aliases) of one library. In this case, separate the names with the equal sign (=). Only one of the library names in the list becomes a virtual directory in the `hdl_libraries` directory.

The library definitions remain in effect until elaboration, after which all library definitions are deleted.

By specifying Verilog and VHDL library names, you can read in multiple Verilog modules and VHDL entities (and VHDL packages) with the same name without overwriting each other. See [Examples](#).

Note: You can type `-lib` or `-library`.

`-netlist`

Reads structural input files when parts of the input design is in the form of a structural netlist. You can read partially structural files provided the structural part of the input design is in the form of structural Verilog-1995 constructs and is contained in separate files from the non-structural (RTL) input.

See [Reading a Partially Structural Design](#) in *Using Encounter RTL Compiler* for detailed information on using the `-netlist` option to read and elaborate a partially structural design.

Note: If this option is specified, all the following options are ignored: `-v1995`, `-v2001`, `-vhdl`, `-sv`.

`-sv`

Specifies that the HDL files conform to SystemVerilog 3.1.a.

`-v1995`

Specifies that the HDL files conform to Verilog-1995.

`-v2001`

Specifies that the HDL files conform to Verilog-2001.

`-vhdl`

Specifies that the HDL files are VHDL files. The `hdl_vhdl_read_version` root attribute value specifies the standard to which the VHDL files conform.

Default: VHDL-1993

Command Reference for Encounter RTL Compiler

Input and Output

Examples

- The following example first loads the `example1.v` file, then the `example2.v` file:

```
rc:/> read_hdl {example1.v example2.v}
```

- The following commands with macro definitions are equivalent:

- `read_hdl -define "A B=4 C"`
 - `read_hdl -define A -define B=4 -define C ...`

- The following command loads a single VHDL file and specifies a single VHDL library.

```
read_hdl -vhdl -library lib1 test1.vhdl
```

- The following commands read structural Verilog files when the design includes RTL (VHDL or Verilog) files:

```
read_hdl file1_bhv.vhdl  
read_hdl file2_bhv.v  
read_hdl -netlist file3_str.v  
elaborate
```

- In the following command, the `-v1995` option is ignored. Both `rtl.v` and `struct.v` are parsed in the structural mode.

```
read_hdl -v1995 rtl.v -netlist struct.v
```

- The following command defines VHDL libraries `lib1`, `lib2` as aliases for `lib3`.

```
read_hdl -vhdl -library lib1=lib2=lib3 test1.vhdl
```

- The following commands read in two Verilog files that each contain a Verilog module with the same name (`compute`) but with different functionality. To store both definitions, the `-lib` option indicates in which library to store the definition.

```
read_hdl -v2001 -library lib1 test_01_1.v  
read_hdl -v2001 -library lib2 test_01_2.v
```

Inspection of the design hierarchy shows:

```
rc:/> ls /hdl_libraries/  
/hdl_libraries:  
./ DP/ DW04/ GB/ STD/ lib2/  
AMBIT/ DW01/ DW05/ GTECH/ SYNERGY/  
CADENCE/ DW02/ DW06/ IEEE/ SYNOPSYS/  
CW/ DW03/ DWARE/ IEEE_SYNERGY/ lib1/  
  
rc:/> ls /hdl_libraries/lib1/architectures/  
/hdl_libraries/lib1/architectures:  
./ compute/  
rc:/> ls /hdl_libraries/lib2/architectures/  
/hdl_libraries/lib2/architectures:  
./ compute/
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Reading a Partially Structural Design](#) in *Using Encounter RTL Compiler*.

Affects this command: [elaborate](#) on page 332

Related command: [read_netlist](#)

Affects this attribute: [hdl_parameter](#)

Affected by these attributes: [hdl_search_path](#)

[hdl_language](#)

[hdl_preserve_dangling_output_nets](#)

[hdl_verilogDefines](#)

read_io_speclist

Refer to [read_io_speclist](#) in Chapter 10, “Design for Test.”

read_netlist

```
read_netlist file_list
    [-top top_module_name]
    [-define macro=value]...
    [-v2001]
```

Reads and elaborates a Verilog 1995 structural netlist when the design does not include behavioral (VHDL or Verilog) modules.

A structural Verilog file contains only structural Verilog-1995 constructs, such as module and gate instances, concurrent assignment statements, references to nets, bit-selects, part-selects, concatenations, and the unary ~ operator. Using the `read_hdl -netlist` command uses less memory and runtime to load a structural file than the `read_hdl` command.

- Use the `read_netlist` command to read and elaborate a design and create a generic netlist that is ready to be synthesized. You do *not* need to use the `elaborate` command
- Use the `read_hdl -netlist` command to read in a design that *includes* behavioral (VHDL or Verilog) files.

Options and Arguments

`-define macro=value`

Defines a Verilog macro with the specified *value*, which is equivalent to the `'define macro value'`.

`file_list`

Specifies the name of the HDL files to load. If several files must be loaded, specify them in a string.

The host directory where the HDL files are looked for is specified via the `hdl_search_path` root attribute.

`-top top_module_name`

Specifies the top-level structural Verilog module to be read and elaborated.

If you do not specify this option and multiple top-level modules are found in the loaded netlist, the tool randomly selects one of them and deletes the remaining top-level modules.

Command Reference for Encounter RTL Compiler

Input and Output

-v2001

Specifies that the netlist files contains Verilog-2001 attributes.
The tool can read in attributes with the following format:

(* attribute[=value] [, attribute[=value] ...] *) ...

The opening and closing parentheses and the stars (*) must be entered literally in the Verilog files.

Related Information

[Reading and Elaborating a Structural Netlist Design](#) and [Reading a Partially Structural Design](#) in *Using Encounter RTL Compiler*.

Related Commands:	<u>read_hdl -netlist</u> <u>write_hdl</u> on page 255
Sets these attributes:	(design) <u>hdl_v2001</u> (instance) <u>hdl_v2001</u> (port) <u>hdl_v2001</u> subdesign <u>hdl_v2001</u> subport) <u>hdl_v2001</u>
Affected by these attributes:	<u>hdl_preserve_dangling_output_nets</u>

read_saif

Refer to [read_saif](#) in Chapter 11, “Low Power Synthesis.”

Command Reference for Encounter RTL Compiler

Input and Output

read_sdc

```
read_sdc file
    [-stop_on_errors] [-no_compress]
    [-mode mode_name]
```

Reads a constraints file in Synopsys Design Constraint (SDC) format into RTL Compiler. RTL Compiler creates a cost group for each clock defined in the file. It does not create false paths between these clocks.

You must first elaborate the design before you can read the design constraints.

If you use the `read_sdc` command for loading a subset of timing constraints that includes native RTL Compiler commands, such as adding exceptions (`path_delays`), the `write_sdc` command will write these exceptions out in the SDC file.

After using the `read_sdc` command, if you make hierarchy changes in RTL Compiler using the `ungroup` or `group` commands, and there are pin specific constraints, then the `write_sdc` command will reflect the change in the hierarchy. For example, if you have constraints on the hierarchy pins you have ungrouped, then the constraints are moved to buffers (that are automatically inserted by RTL Compiler when ungrouping). The SDC file will have constraints reflecting these buffers.

Unsupported Constraints

Not all SDCs are supported. For those that are not supported, RTL Compiler will issue a warning message but store them for output for the `write_sdc` command. RTL Compiler will only store the SDCs and not manipulate any data with them.

The following SDCs are not supported:

```
set_max_area
set_propagated_clock
set_scan_style
set_signal_type
set_test_methodology
set_wire_load_min_block_size
get_references
get_reference
set_connection_class
set_critical_range
set_fix_multiple_port_nets
set_local_link_library
```

Command Reference for Encounter RTL Compiler

Input and Output

Options and Arguments

<i>file</i>	Specifies the name constraints file to read. You can also specify a file that was compressed with gzip (.gz extension).
-mode mode_name	Reads mode specific constraints for a design.
-no_compress	Turns off advanced compression and compression of exceptions for the remainder of the session.
-stop_on_errors	Stops reading the remainder of the script if an error is encountered during reading of the SDC file.

Related Information

[Applying Design Constraints in Using Encounter RTL Compiler](#)

[Performing Multi-Mode Timing Analysis in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

Affects this command: [synthesize](#) on page 348
[create_mode](#) on page 287

Related command: [write_sdc](#) on page 265

read_spef

Refer to [read_spef](#) in [Chapter 9, “Physical.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

read_tcf

Refer to [read_tcf](#) in [Chapter 11, “Low Power Synthesis.”](#)

read_vcd

Refer to [read_vcd](#) in Chapter 11, “Low Power Synthesis.”

Command Reference for Encounter RTL Compiler

Input and Output

restore_design

```
restore_design
  -db_dir string -design_name design
  [-def file] [-worst_corner string]
```

Loads the database written out by the Encounter® tool in the RTL Compiler tool.

Options and Arguments

<code>-db_dir <i>string</i></code>	Specifies the path to the Encounter database directory.
<code>-def <i>file</i></code>	Specifies the path to the DEF file. If this option is not specified, the tool searches for a <i>design.def</i> or <i>design.def.gz</i> file in the Encounter database directory. If neither file is found, an error message is issued.
<code>-design_name <i>design</i></code>	Specifies the name of the design. The design name is the base filename for the output files generated by Encounter tool.
<code>-worst_corner <i>string</i></code>	Specifies the worst case delay corner of all corners defined in the <i>viewdefinitions.tcl</i> file. This option is required for multi-mode multi-corner designs. Note: The libraries and captable for the worst corner will be loaded in the RTL Compiler tool.

Example

The following command reads the Encounter database from the *fe_db_dat* directory, specifies that the name of the design is *test*. The assumptions are that a *test.def* or *test.def.gz* file is part of the *fe_db_dat* directory and that the design is a non-MMMC design.

```
restore_design -db_dir fe_db_1.dat -design test
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

Affects this command: [synthesize on page 348](#)

split_db

```
split_db  
  {input_db_file | -from_tcl string}  
  -script file [-to_file file]
```

Reads a database with setup information and writes out the setup information in a setup script and the netlist information into a setup-free database.

Options and Arguments

<i>-from_tcl string</i>	Specifies to read the specified Tcl object.
<i>-script file</i>	Specifies the name of the script to be written. If this option is specified without the <i>-to_file</i> option, the tool writes both the setup and the database to the script. This ensures that the Tcl script and its database are always in sync.
	Note: Storing the database in the script takes more memory and results in a larger file.
<i>-to_file file</i>	Specifies the name of the new database file.
<i>input_db_file</i>	Specifies the name of the database file to be read.

Related Information

[Using the RTL Compiler Database](#) in *Using Encounter RTL Compiler*

Related commands:

read_db on page 195
write_db on page 221

write_atpg

Refer to [write_atpg](#) in Chapter 10, “Design for Test.”

Command Reference for Encounter RTL Compiler

Input and Output

write_bsdl

Refer to [write_bsdl](#) in Chapter 10, “Design for Test.”

write_compression_macro

Refer to [write_compression_macro](#) in [Chapter 10, “Design for Test.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_cpf

Refer to [write_cpf](#) in [Chapter 12, “Advanced Low Power Synthesis.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_db

```
write_db
  [-to_file db_file]
  [-all_root_attributes | -no_root_attributes]
  [-script file] [design] [-quiet] [-verbose]
```

Writes the netlist to a database file or returns a Tcl list. Root attributes with non-default settings can be included in the database or written to a script. By default, only root attributes affecting the QOR are written out.

Options and Arguments

-all_root_attributes

Writes out all the root attributes with non-default settings.

design Specifies the design for which to write out the database information.

If omitted, the design defaults to the current design.

-no_root_attributes

Prevents writing out of the root attributes with non-default settings.

You cannot specify this option with the **-script** option.

-quiet Suppresses any warning messages. Error messages are printed.

-script file Includes all root attribute settings in the specified script.

If this option is omitted, this information is included in the database file.

If this option is specified without the **-to_file** option, the tool writes both the setup and the database to the script. This ensures that the Tcl script and its database are always in sync.

Note: Storing the database in the script takes more memory and results in a larger file.

-to_file db_file Specifies the name of the database file to be written.

By default, the command returns a Tcl list.

-verbose Enables verbose output while writing out the database file.

Command Reference for Encounter RTL Compiler

Input and Output

Example

In the following script, the variable `db` is defined for the Tcl object written out by the `write_db` command:

```
set db [write_db /designs/test]
...
read_db -from_tcl $db
```

Related Information

[Using the RTL Compiler Database](#) in *Using Encounter RTL Compiler*

Related commands:

- [define_attribute](#) on page 983
- [read_db](#) on page 195
- [split_db](#) on page 216
- [write_design](#) on page 224

write_def

See [write_def](#) on page 566 in [Chapter 9, “Physical.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_design

```
write_design
  [-basename string] [-gzip_files] [-tcf]
  [-encounter] [design]
```

Generates all the files needed to reload the session in RTL Compiler (for example, .g, .v, and .tcl files). If you want to generate all the files that are need to loaded in both a RTL Compiler and Encounter® session, use the -encounter option.

Note: When you use this command on a design that is not fully mapped, for example after elaborate or synthesize -to_generic, and then reload and map the design, the final area and timing results may differ from the results obtained in a single synthesis session.

When you write out the design after spatial optimization (synthesize -spatial), an encrypted file (*basename.spl.etcf*) is written. To reload this file, use the decrypt command.

Options and Arguments

<code>-basename <i>string</i></code>	Specifies the path and basename for the generated files.
<code><i>design</i></code>	Specifies the top-level design in RTL Compiler.
<code>-encounter</code>	Generates the additional files needed for Encounter.
<code>-gzip_files</code>	Compresses the generated files in gzip format.
<code>-tcf</code>	Specifies to write out a TCF containing the asserted switching activities of the pins in the design.

Example

- The following example writes both the RTL Compiler and Encounter files as well as specifies the path and basename to be test/top:

```
rc:/> write_design -encounter -basename test/top
unix> ls /home/mydir/test
top.conf
top.g
top.rc_setup.tcl
top.v
top.enc_setup.tcl
top.mode
top.sdc
top.determinate.tcl
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Generating Design and Session Information in *Using Encounter RTL Compiler*](#)

[Saving and Restoring a Session in RTL Compiler in *Using Encounter RTL Compiler*](#)

write_dft_abstract_model

Refer to [write_dft_abstract_model](#) in Chapter 10, “Design for Test.”

Command Reference for Encounter RTL Compiler

Input and Output

write_do_ccd

```
write_do_ccd {compare_sdc | generate | propagate | validate}
```

Translates RTL Compiler settings to Conformal's Constraint Designer (CCD) commands for the *Validate* and *Generate* flows. In the *Validate* flow, by default the command compares the SDC to the RTL.

Options and Arguments

compare_sdc	Generates a dofile to compare two SDC files.
generate	Generates a dofile for the <i>Generate</i> flow.
propagate	Generates a dofile to create a chip-level SDC file.
validate	Generates a dofile for the <i>Validate</i> flow.

Related Information

[Interfacing with Encounter Conformal Constraint Designer](#) in *Interfacing Between RTL Compiler and Conformal*

Related commands:	write_do_ccd compare_sdc on page 228
	write_do_ccd generate on page 230
	write_do_ccd propagate on page 232
	write_do_ccd validate on page 234

write_do_ccd compare_sdc

```
write_do_ccd compare_sdc
  [-design string] [-netlist file]
  -golden_sdc file -revised_sdc file
  [-pre_load script] [-pre_exit script]
  [-no_exit] [-logfile file] [-detail] [> file]
```

Writes a dofile for the Encounter® Conformal® Constraint Designer (CCD) to compare two SDC files and report any differences between the two files.

Options and Arguments

-design <i>string</i>	Specifies the top-level design in RTL Compiler.
-detail	Requests a detailed comparison report.
<i>file</i>	Specifies the file to which the report must be written.
-golden_sdc <i>file</i>	Specifies the UNIX path to the golden SDC file.
-logfile <i>file</i>	Specifies the name of the CCD logfile. You must specify the UNIX path to the file.
-netlist <i>file</i>	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
-no_exit	Suppresses the EXIT command at the end of the dofile.
-pre_exit <i>string</i>	Specifies the name of the dofile (script) that must be sourced before the CCD EXIT command.
-pre_read <i>string</i>	Specifies the name of dofile (script) that must be sourced before the libraries and the design are read.
-revised_sdc <i>file</i>	Specifies the UNIX path to the revised SDC file.

Example

The following command compares the test.sdc and revised.sdc files.

```
write_do_ccd compare_sdc -golden_sdc test.sdc -revised_sdc revised.sdc
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Comparing SDC Constraint Files](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Related command: [compare_sdc](#) on page 373

write_do_ccd generate

```
write_do_ccd generate
    [-design string] [-netlist string]
    -in_sdc files [-out_sdc file]
    [-slack integer] [-report file]
    [-dfpgen | -fpfgen | -trv]
    [-pre_load script] [-pre_exit script]
    [-no_exit] [-logfile file] [> file]
```

Writes a dofile for the Encounter® Conformal® Constraint Designer (CCD) for the *Generate* flow, which generates additional false paths based on critical path timing reports.

Options and Arguments

<code>-design <i>string</i></code>	Specifies the top-level design in RTL Compiler.
<code>file</code>	Redirects all the output to the specified file.
<code>-dfpgen</code>	Generates a dofile for the directed false path flow.
<code>-fpfgen</code>	Generates a dofile for the false path flow.
<code>-in_sdc <i>files</i></code>	Specifies the list of SDC files to load.
<code>-logfile <i>file</i></code>	Specifies the name of the CCD logfile.
<code>-netlist <i>string</i></code>	Specifies the UNIX path to the netlist. This option compares the SDC to the specified netlist instead of the RTL.
<code>-no_exit</code>	Suppresses the <code>exit</code> command at the end of the dofile.
<code>-out_sdc <i>file</i></code>	Specifies the filename to which the identified false paths will be written. <i>Default:</i> <code>cfp.sdc</code>
<code>-pre_exit <i>string</i></code>	Specifies the name of the dofile (script) that must be sourced before the CCD exit command.
<code>-pre_read <i>string</i></code>	Specifies the name of dofile (script) that must be sourced before the libraries and the design are read.
<code>-report <i>file</i></code>	Specifies the name of the timing report file to be generated. The report will be in CCD format.
<code>-slack <i>integer</i></code>	Specifies the slack value in picoseconds. Only paths below this slack value will be used to generate the timing report. This option should be used with the <code>-report</code> option.

Command Reference for Encounter RTL Compiler Input and Output

-trv Generates a dofile for the timing report validation flow.

Related Information

Using the Generate Flow with Dofiles in Interfacing Between Encounter RTL Compiler and Encounter Conformal

Affected by this attribute: wcccd_threshold_percentage

write_do_ccd propagate

```
write_do_ccd propagate
  [-design design] [-netlist string]
  -block_sdc string [-glue_sdc string]
  [-partial_chip_sdc string]
  [-out_sdc string]
  [-rule_instance_file string]
  [-rule_instance_template string]
  [-pre_load script] [-pre_exit script]
  [-no_exit] [-logfile string] [> file]
```

Generates a dofile for the Encounter® Conformal® Constraint Designer (CCD) to propagate block-level constraints to the top-level and integrate them with the glue constraints to generate a chip-level SDC file.

Options and Arguments

-block_sdc <i>string</i>	Specifies a list of block names with their associated block-level SDC files in the following format: { <i>block_name</i> <i>block_sdc_file</i> }...{}
-design <i>string</i>	Specifies the top-level design in RTL Compiler.
<i>file</i>	Specifies the file to which the report must be written.
-glue_sdc <i>string</i>	Specifies the name of a glue SDC file. This file contains a set of constraints for the top-level module only (without covering any block-level constraints).
-logfile <i>string</i>	Creates a separate CCD logfile. You must specify the UNIX path to the file.
-netlist <i>string</i>	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
-no_exit	Suppresses the exit command at the end of the dofile.
-out_sdc <i>string</i>	Specifies the name of the constraints file that is generated after propagation and integration of the block and glue constraints. <i>Default:</i> chip.sdc
-partial_chip_sdc <i>string</i>	Specifies the name of the partial SDC file that corresponds to the top-level of the design.

Command Reference for Encounter RTL Compiler

Input and Output

<code>-pre_exit string</code>	Specifies the name of the dofile (script) that must be sourced before the CCD exit command.
<code>-pre_read string</code>	Specifies the name of dofile (script) that must be sourced before the libraries and the design are read.
<code>-rule_instance_file string</code>	Specifies the name of the file which defines the rule instances for the Encounter® Conformal® Constraint Designer (CCD) tool.
<code>-rule_instance_template string</code>	<p>Creates a template integration rule instances file.</p> <p>You can modify this file according to the design. To use this file as input for the Encounter® Conformal® Constraint Designer (CCD) tool, specify the file as value of the <code>-rule_instance_file</code> option.</p>

Example

- The following command creates a do file to propagate the block-level SDC files `i1.sdc` and `i2.sdc` to the top level.

```
rc:/> write_do_ccd propagate -block_sdc {{i1 i1.sdc} {i2 i2.sdc}} \
    -out_sdc ./my_chip.sdc rule_instance_file my_rules -logfile ccd.log
```

The generated dofile will be similar to:

```
read library -statetable -liberty ./slow.lib
add search path -design .
read design -verilog ./til1.v -lastmod -noelab
elaborate design
dofile ./my_rules
read hierarchical sdc \
-sdc_design i1 i1.sdc \
-sdc_design i2 i2.sdc
set system mode verify
integrate -all ./chip.sdc -replace
report rule check
report environment
```

write_do_ccd validate

```
write_do_ccd validate
    [-design string] [-netlist string]
    -sdc string
    [-init_sequence_file string]
    [-pre_load script] [-pre_exit script]
    [-no_exit] [-logfile string] [> file]
```

Writes a Conformal Constraint Designer (CCD) dofile for the *Validate* flow, which validates the constraints and false path exceptions.

Options and Arguments

<code>-design <i>string</i></code>	Specifies the top-level design in RTL Compiler.
<code><i>file</i></code>	Redirects all the output to the specified file.
<code>-init_sequence_file <i>string</i></code>	Specifies the UNIX path to the initialization sequence file for multi-cycle path validation.
<code>-logfile <i>string</i></code>	Specifies the name of the CCD logfile.
<code>-netlist <i>string</i></code>	Specifies the UNIX path to the netlist. This option compares the SDC to the specified netlist instead of the RTL.
<code>-no_exit</code>	Suppresses the <code>exit</code> command at the end of the dofile.
<code>-pre_exit <i>string</i></code>	Specifies the name of the dofile (script) that must be sourced before the CCD exit command.
<code>-pre_read <i>string</i></code>	Specifies the name of dofile (script) that must be sourced before the libraries and the design are read.
<code>-sdc <i>string</i></code>	Specifies the list of SDC files.

Related Information

[Using the Validate Flow with Dofiles in Interfacing Between Encounter RTL Compiler and Encounter Conformal](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_do_clp

```
write_do_clp
  [-design design] [-netlist string ]
  [-env_var string]
  [-add_iso_cell string] [-clp_out_report string]
  [-ignore_ls_htol] [-verbose]
  [-pre_load script] [-pre_exit script]
  [-no_exit] [-tmp_dir string] [-logfile file] [> file]
```

Writes the required dofile for Conformal Low Power (CLP).

Note: This command will issue an error and will not proceed if you have multiple Common Power Format (CPF) files.

For more information on the Low Power Rule (CLP) Checks, refer to the Encounter® Conformal® Low Power Reference Manual.

Options and Arguments

-add_iso_cell *string*

Specifies the standard cells that CLP should recognize as isolation cells.

-clp_out_report *string*

Writes the output of the report rule check Encounter® Conformal® Equivalence Checking command to this specified file.

-design *design*

Specifies the top-level design in RTL Compiler.

-env_var *string*

Specifies the names and values of UNIX environment variables to be used in the library, design, and logfile names in the generated dofile.

file

Redirects all the output to the specified file.

-ignore_ls_htol

Indicates whether to ignore the high to low level shifter check. If this option is specified, the following CLP directive will be added to the dofile:

```
set lowpower option -ignore_high_to_low
```

-logfile *file*

Specifies the name of the CLP logfile.

Command Reference for Encounter RTL Compiler

Input and Output

-netlist <i>path</i>	Specifies the UNIX path that contains the netlist containing all the design's low power features (for example, level shifters, isolation cells and SRPG flops).
	<i>Default:</i> RTL
-no_exit	Indicates whether to skip the <code>exit</code> command at the end of the dofile. If this option is specified, the following command will be omitted from the dofile:
	<code>exit -force</code>
-pre_exit <i>string</i>	Specifies the name of the dofile (script) that must be sourced before the CLP <code>exit</code> command.
-pre_read <i>string</i>	Specifies the name of dofile (script) that must be sourced before the libraries and the design are read.
-tmp_dir <i>string</i>	Specifies the name of the directory to which the generated files must be written. Its contents will be CLP native commands.
	<i>Default:</i> <code>RC_CLP_design_name_out.do</code>
-verbose	Indicates whether the generated dofile will be verbose. If this option is specified, the <code>report rule check</code> command should write out the intermediate dofile for CLP and then include that file.

Example

- The following Encounter Conformal Equivalence Checking commands is an example of a CLP dofile:

```
set log file log_file_name -replace
set lowpower option -netlist_style logical

read library -statetable -liberty bn65lplvt_121a/tcbn65lplvtwcl0d90d72.lib \
read design -verilog -sensitive netlist.v

read cpf file cpf_file_name

analyze power domain
rep rule check ISO* LSH* RET* -verbose
exit -force
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Interfacing with Conformal Low Power](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Affected by these attributes: [wclp_lib_statetable](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_lec

```
write_lec [-top string]
           [-golden_design string] [-revised_design string]
           [-cpf_golden file] [-cpf_revised file]
           [-sim_lib string] [-sim_plus_liberty]
           [-logfile string] [-env_var string]
           [-pre_read string] [-pre_compare string]
           [-pre_exit string]
           [-hier | -flat] [-no_exit]
           [-save_session string] [-tmp_dir string]
           [-verbose] [> file]
```

Translates RTL Compiler settings to Encounter® Conformal® Logical Equivalence Checking commands.

This command works with the Common Power Format (CPF) flow: if it detects a CPF file then it will output this information to the Conformal LEC dofile.

Options and Arguments

<code>-cpf_golden <i>file</i></code>	Specifies the name of the golden (original) file. If omitted, the file will be inferred from the <code>cpf_files</code> design attribute. Note: This option only applies to the CPF flow.
<code>-cpf_revised <i>file</i></code>	Specifies the name of the revised CPF file. If you omit this option, the file specified for the <code>-cpf_golden</code> option will be used. If you expect name changes in the CPF design objects, you must generate this file using the <code>write_cpf</code> command. Note: This option only applies to the CPF flow.
<code>-env_var <i>string</i></code>	Specifies the names and values of UNIX environment variables to be used in library, design, and log filenames in the dofile.
<code><i>file</i></code>	Redirects all the output to the specified file.
<code>-flat</code>	Performs a flattened Conformal LEC comparison.
<code>-golden_design <i>string</i></code>	Specifies the UNIX path to an alternative golden design.

Command Reference for Encounter RTL Compiler

Input and Output

If the file was loaded into RTL Compiler (using either `read_hdl` or `read_netlist`), the tool knows the language format of the file.

Otherwise, the tool assumes that the format of the file is Verilog-1995.

<code>-hier</code>	Performs a hierarchical Conformal LEC comparison.
<code>-logfile string</code>	Specifies the name of the Conformal LEC logfile.
<code>-no_exit</code>	Does not add the <code>exit</code> command to the end of the dofile.
<code>-pre_compare string</code>	Specifies an extra dofile that must be sourced from the dofile before the LEC <code>compare</code> command.
<code>-pre_exit string</code>	Specifies an extra dofile that must be sourced from the dofile before the LEC <code>exit</code> command.
<code>-pre_read string</code>	Specifies an extra dofile that must be sourced from the dofile before the library and design are read.
<code>-revised_design string</code>	Specifies the UNIX path to the revised design.
<code>-save_session string</code>	Specifies the filename to save the LEC session.
<code>-sim_lib string</code>	Specifies the simulation library in Verilog 1995.
<code>-sim_plus_liberty</code>	Specifies that the simulation library is an addition to the synthesis library.
<code>-tmp_dir string</code>	Specifies the name of the directory to which the generated files must be written.
<code>-top string</code>	Specifies the name of the top-level design in RTL Compiler.
<code>-verbose</code>	Generates a dofile with verbose reporting.

Example

- The following command will source the `extra_settings.do` dofile before the `compare` command in the `sample.do` dofile.

```
write_dolec -revised revised.v -pre_compare extra_settings.do > sample.do
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Interfacing with Encounter Conformal Logical Equivalence Checker](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

[Performing a Low Power Equivalence Check](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

[More About the write_lec_dofile](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Related command: [write_cpf](#) on page 930

Affected by these attributes: [boundary_optimize_invert_hier_pins](#)

[wlec_add_noblock_box_retime_subdesign](#)

[wlec_analyze_abort](#)

[wlec_analyze_setup](#)

[wlec_auto_analyze](#)

[wlec_compare_threads](#)

[wlec_cut_point](#)

[wlec_hier_comp_threshold](#)

[wlec_lib_statetable](#)

[wlec_set_cdn_synth_root](#)

[wlec_uniquify](#)

[wlec_use_lec_model](#)

write_do_verify cdc

```
write_do_verify cdc {-categorize | -validate}
    -sdc string
    [-design string] [-no_exit]
    [-logfile string] [> file]
```

Generates a dofile for Encounter® Conformal® Extended Checks to perform clock domain crossing checks on clock domain crossings in either the Categorization or Validation flow.

- In the Categorization flow, no synchronization rules are defined. RTL Compiler automatically identifies and categorizes the clock domain crossing paths.
- In the Validation flow, you define the synchronization rules that specify the valid synchronization structures in the design.

Options and Arguments

<code>-categorize</code>	Specifies to generate a dofile for the <i>Categorization</i> flow.
<code>-design <i>string</i></code>	Specifies the name of the top-level design in RTL Compiler.
<code><i>file</i></code>	Specifies a specific dofile filename.
<code>-logfile <i>string</i></code>	Specifies a specific logfile name.
<code>-no_exit</code>	Suppresses the <code>exit</code> command in the dofile.
<code>-sdc <i>string</i></code>	Specifies a list of the SDC files.
<code>-validate</code>	Specifies to generate a dofile for the <i>Validation</i> flow.

Examples

- The following example writes a dofile for the Categorization flow:

```
write_do_verify cdc -sdc /home/test/general.sdc -logfile my.log \
    -categorize
```

- The following example writes a dofile for the Validate flow:

```
write_do_verify cdc -sdc /home/test/general.sdc -logfile my.log \
    -validate
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

[Interfacing with Encounter Conformal Extended Checks](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Affected by these attributes:

wcdc_clock_dom_comb_propagation
wcdc_synchronizer_type

Command Reference for Encounter RTL Compiler

Input and Output

write_encounter

```
write_encounter design [-basename string]
                      [-tcf] [-reference_config_file config_file]
                      [-gzip_files] [-ignore_scan_chains] [-ignore_msv]
                      [-floorplan string] [-lef lef_files] [design]
```

Writes Encounter® input files to a single directory. The command will only convert library domains into power domains for Encounter if power domains exist in RTL Compiler. If power domains do not exist, the `-ignore_msv` option is implied. The command also supports the Common Power Format (CPF) files by directly passing them to Encounter.

The generated files are all required Encounter input files and include the following files:

- Netlist (.v)
- Encounter configuration file (.conf),
- SDC constraints (.sdc)
- Tcl script (.enc_setup.tcl)
- Mode file (.mode)
- Scan DEF file (.scan.def)
- MSV-related files (.msv.tcl, .msv.vsf)
- Multiple timing mode (.mmode.tcl)
- Updated CPF file (for a CPF-based flow)

The `.enc_setup.tcl` file can simultaneously load all the necessary Encounter data in an Encounter session. This eliminates the need to load each of the necessary files sequentially.

The `.mode` file contains all the Encounter `setMode` settings. For example, the file would contain the `setAnalysisMode` and `setPlaceMode` settings.

The full DEF file that is outputted is the exact same DEF file that was loaded or generated by `synthesize -to_placed`. However, RTL Compiler generates the information for the Scan DEF file (.scan.def).

The updated CPF file contains the power intent of the golden CPF but with updated references to the design objects.

Note: The MSV (multiple supply voltage) library domain setup commands require Encounter version 4.2 or later. Multiple timing mode is supported in Encounter version 5.2 or later.

Command Reference for Encounter RTL Compiler

Input and Output

Options and Arguments

<code>-basename string</code>	Specifies the directory path name and base filename for the output data. The default directory is <code>./rc_enc_des</code> and the default filename without the extension is <code>rc</code> .
<code>design</code>	Specifies a particular design for which to write out information. Only one design can be specified at a time.
<code>-floorplan string</code>	Specifies the extension for the file containing the floorplan. The valid extensions are: <code>.def</code> —DEF <code>.pde</code> —PDEF <code>.fp</code> —Encounter floorplan
<code>-gzip_files</code>	Compresses the netlist and constraints in <code>.gz</code> format. The floorplan, if one was read, will be untouched. That is, if it was read in uncompressed, it will be outputted uncompressed and vice versa. Note: Since Global Timing Debug (in Encounter Timing System) does not handle compressed SDC files, the <code>write_encounter</code> command will not compress SDC files.
<code>-ignore_scan_chains</code>	If specified, the scan DEF file will not be written and the scan reorder directives will not be included in the setup file.
<code>-ignore_msv</code>	If specified, the MSV setup file and the shifter table file will not be written out. This option is useful if the library domains in RTL Compiler are not being used for modeling power domains.
<code>-lef lef_files</code>	Specifies a particular physical library or libraries to use. The physical libraries will have the <code>.lef</code> extension. The contents of the LEF library that was specified with the <code>lef_library</code> attribute will be used if this option is not specified.
<code>-reference_config_file config_file</code>	Specifies a reference Encounter configuration file to use as a template for the generated configuration file.
<code>-tcf</code>	Specifies to write out a TCF containing the asserted switching activities of the pins in the design.

Examples

- The following example writes all the Encounter input files for the test07 design to the directory TEST. The basename for the files are specified to be test1.

```
rc:/> write_encounter design test07 -basename TEST/test1

unix> ls TEST/
test1.conf    test1.enc_setup.tcl    test1.mode    test1.sdc    test1.v
```

- The following example compresses the netlist and constraints with the `-gzip_files` option:

```
rc:/> write_encounter design -gzip_files

unix> ls rc_enc_des
rc.conf    rc.def    rc.enc_setup.tcl    rc.mode    rc.sdc.gz    rc.v.gz
```

Related Information

[Performing Multi-Mode Timing Analysis in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

[Export to Place and Route in The Multiple Supply Voltage Flow in Low Power in Encounter RTL Compiler.](#)

Related commands: [create mode](#) on page 287
[read encounter](#) on page 200

write_et_atpg

Refer to [write_et_atpg](#) in [Chapter 10, “Design for Test.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_et_bsv

Refer to [write_et_bsv](#) in [Chapter 10, “Design for Test.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_et_dfa

Refer to [write_et_dfa](#) in Chapter 10, “Design for Test.”

Command Reference for Encounter RTL Compiler

Input and Output

write_et_ibist

Refer to [write_et_ibist](#) in Chapter 10, “Design for Test.”

write_et_mbist

Refer to [write_et_mbist](#) in [Chapter 10, “Design for Test.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_et_rrfa

Refer to [write_et_rrfa](#) in [Chapter 10, “Design for Test.”](#)

write_ets

```
write_ets [-default] [-ocv]
           [-pre_include string] [-post_include string]
           [-netlist string]
           [-sdc string] [-sdf string] [-spef string]
           [> file]
```

Generates an Encounter® Timing System (ETS) run script.

Options and Arguments

-default	Generates a simple ETS run script. The script will contain the following ETS commands: <code>read_lib</code> , <code>read_verilog</code> , <code>set_top_module</code> , <code>read_sdc</code> , and <code>report_timing</code> .
file	Redirects the output to the specified file.
-netlist string	Specifies the UNIX path of the file containing the gate-level netlist.
-ocv	Adds an extra <code>set_timing_derate</code> command into the ETS run script. This option can only be specified with the <code>-sdf</code> option.
-post_include string	Specifies the UNIX path of the include file that contains ETS commands that need to be added after the <code>report_timing</code> command in the run file generated by <code>write_ets</code> .
-pre_include string	Specifies the UNIX path of the include file that contains ETS commands that need to be added before the <code>report_timing</code> command in the run file generated by <code>write_ets</code> .
-sdc string	Specifies the UNIX path of the SDC file.
-sdf string	Specifies the UNIX path of the SDF file. If this option is specified, the <code>read_sdf</code> , <code>set_analysis_mode</code> , and <code>set_op_cond</code> commands will be added to the ETS run script.
-spef string	Specifies the UNIX path of the SPEF file. If this option is specified, the <code>read_spef</code> , <code>set_analysis_mode</code> , and <code>set_op_cond</code> commands will be added to the ETS run script.

write_ett

```
write_ett
  [-strict | -dc | -ett]
  [-version {1.1|1.3|1.4}]
  [design] [> file]
```

Generates constraints for Encounter® True Time.

Some constraints (such as `set_input_delay`, `set_output_delay`, and so on) are written out in SDC (Synopsys Design Constraints) format while others (such as `set_false_path`, `set_disable_timing`) are written out in Encounter test format.

Options and Arguments

<i>design</i>	Specifies the design for which the constraints must be generated.
<i>file</i>	Specifies the name of the file to which the constraints must be written.
<code>-dc</code>	Writes the constraints that are DC and PT compatible, which means that commands not listed in the SDC specification may be written out.
<code>-ett</code>	Writes an Encounter True Time Clock Constraints file.
<code>-strict</code>	Writes only constraints in SDC format.
<code>-version {1.1 1.3 1.4}</code>	Specifies the SDC version to use to write SDC constraints.

Command Reference for Encounter RTL Compiler

Input and Output

write_forward_saif

Refer to [write_forward_saif](#) in Chapter 11, “Low Power Synthesis.”

Command Reference for Encounter RTL Compiler

Input and Output

write_hdl

```
write_hdl {design|subdesign}...
    [-suffix string]
    [-abstract] [-generic] [-depth integer]
    [-equation] [-lec] [-v2001] [> file]
```

Generates one of the following design implementations in Verilog format:

- A structural netlist using generic logic
- A structural netlist using mapped logic

You can automatically read in or write out a gzip compressed Verilog file. For example:

```
read_hdl sample.v.gz
write_hdl > sample.v.gz
```

Options and Arguments

-abstract	Generates an empty top-level Verilog module definition of the specified design or subdesign that defines the I/O pins and bit-width for all top-level functional and scan-related ports in the design or subdesign. This empty module description is further referred to as <i>logic abstract model</i> .
-depth <i>integer</i>	Specifies the number of hierarchy levels to be written out, starting from the top level. A value of 0, writes out only the top-level module. <i>Default:</i> infinite
{ <i>design</i> <i>subdesign</i> }	Specifies the design or subdesign for which the design implementation must be generated.
-equation	Writes out a logic equation in an assign statement for each Verilog primitive gate.
<i>file</i>	Specifies the file to which the output must be written. <i>Default:</i> Output is written to the screen.
-generic	Generates an unoptimized generic logic implementation of the design that uses the generic logic gates specified within the Verilog language.

Command Reference for Encounter RTL Compiler

Input and Output

Any parts of the design that are mapped will be unmapped for the `write_hdl` command without affecting the design in memory.

Once the `synthesize` command has been run, you cannot recover the version of the design that was generated using this option prior to synthesis.

-lec	Generates an <i>intermediate</i> netlist with additional information to facilitate formal verification with Encounter® Conformal® Equivalence Checking. See Related Information for more information on when to write out the intermediate netlist.
-suffix	Specifies the string to be appended to the name of all defined modules in the generated netlist.
-v2001	Writes out the Verilog-2001 attributes stored with instances, ports and subports in the design.

Examples

- The following example writes out the logic abstract model definition of design `test`:

```
rc:/> write_hdl -abstract
// Generated by Cadence RTL Compiler-D (RC) version
module test(in1, in2, out1, out2, clk1, clk2, clk3, sel, se2);
    input [3:0] in1;
    input [7:0] in2;
    input clk1, clk2, clk3, sel, se2;
    output [3:0] out1;
    output [7:0] out2;
endmodule
```

- The following example writes out the design as generic logic regardless of its current mapped state:

```
rc:/> write_hdl -generic > design_rtl.v
```

- You can write out a netlist for a specific module. For example, the following commands writes out the `middle` module:

```
rc:/> set_attr unresolved true [get_attr instance [get_attr subdesign bottom]]
rc:/> write_hdl [find / -subdesign middle]
```

- The following example writes out a design that instantiates cells from the target technology library reflecting the current state of the design (mapped state):

```
rc:/> read_hdl design.v
rc:/> ...
rc:/> synthesize -to_mapped
rc:/> ...
rc:/> write_hdl
```

Command Reference for Encounter RTL Compiler

Input and Output

- The following example replaces each Verilog primitive gate by an equivalent Verilog assign statement:

```
rc:/> write_hdl -equation design.v
```

- The following example writes out the design as generic logic regardless of its current mapped state:

```
rc:/> write_hdl -generic > design_rtl.v
```

Related Information

[Writing Out the Design Netlist](#) in *Using Encounter RTL Compiler*.

[Creating a Logic Abstract Model](#) in *Design for Test in Encounter RTL Compiler*

[When to Write out Intermediate Netlist](#) in *Interfacing Between Encounter RTL Compiler and Encounter Conformal*

Affected by these commands:	elaborate on page 332 synthesize on page 348
Affected by these attributes:	optimize_merge_flops optimize_merge_latches optimize_merge_seq write_sv_port_wrapper write_vlog_bit_blast_bus_connections write_vlog_bit_blast_constants write_vlog_bit_blast_mapped_ports write_vlog_bit_blast_tech_cell write_vlog_convert_onebit_vector_to_scalar write_vlog_declare_wires write_vlog_empty_module_for_logic_abstract write_vlog_line_wrap_limit write_vlog_no_negative_index write_vlog_port_association_style write_vlog_preserve_net_name write_vlog_top_module_first

Command Reference for Encounter RTL Compiler

Input and Output

write_vlog_unconnected_port_style
write_vlog_wor_wand

write_io_speclist

Refer to [write_io_speclist](#) in [Chapter 10, “Design for Test.”](#)

write_saif

Refer to [write_saif](#) in [Chapter 11, “Low Power Synthesis.”](#)

write_scandef

Refer to [write_scandef](#) in [Chapter 10, “Design for Test.”](#)

write_script

```
write_script [-hdl]
    [-analyze_all_scan_chains [-dont_overlay_segments]]
    [design] [> file]
```

Generates a script that contains the timing for all modes and the design rule constraints of the design. If you used DFT functionality, the script will also contain any test constraints that were applied, as well as any objects that were created in the design as a result of inserting DFT logic such as boundary scan, building the fullscan chains, and inserting scan chain compression logic.

The resulting script can subsequently be used to examine the current design constraints, or it can be read back into RTL Compiler to perform analysis or optimization at a later time.

The `write_script` command can also compress the output using the `gzip` (`.gz` extension).

The script contains the following:

- The attributes connected with the `wire_load` models
- Clock objects and their reference to the pins of the design blocks
- `External_delay` on all inputs and outputs
- Timing exceptions
- `max_fanout / max_capacitance` and similar design rule constraints applied
- All user defined attributes that were created with the `define_attribute` command

The script can also include DFT constraints or commands, such as:

- DFT constraints created (or tool inferred) using any of the following:
`define_dft shift_enable, define_dft test_mode, define_dft test_clock, define_dft scan_chain`
- DFT constraints created with `set_attribute dft_dont_scan`
- DFT objects created by the user or by the tool with `set_attribute user_defined` and `set_attribute dft_auto_created`
- `check_dft_rules`

Note: The `write` command writes out only the design itself while the `write_script` command writes out the constraints for the design.

Command Reference for Encounter RTL Compiler

Input and Output

Options and Arguments

-analyze_all_scan_chains

Writes out all chains in the dft/report/
actual_scan_chains directory using the following notation:

```
define_dft scan_chain -name name... -sdo sdo -analyze
```

When running the script in a new RTL Compiler session, the RC-DFT engine analyzes the existing scan chains (traces the connectivity of the chains) and restores this information into the dft/report/actual_scan_chains directory.

design

Specifies the name of the design for which to write a script.

-dont_overlay_segments

Adds the -dont_overlay option to the scan chains it is writing out with the -analyze option.

Note: You can only specify this option when you specify the -analyze_all_scan_chains option.

file

Specifies the name of the file to which to write the constraints.

-hdl

Writes out the architecture/entity filename information to the output file.

Examples

- The following example saves the design and its constraints:

```
rc:/> write_hdl > mapped.v
rc:/> write_script > mapped.g
```

The design and script is subsequently read into another RTL Compiler session. You must specify any .lib, LEF, or cap table files: these files are process specific as opposed to design specific and therefore are not automatically loaded.

```
rc:/> set_attribute library areid.lib
rc:/> set_attribute lef_library areid.lef
rc:/> set_attribute cap_table_file areid.cap
rc:/> read mapped.v
rc:/> elaborate
rc:/> source mapped.g
```

- The following example automatically compresses the output file using the .gz extension:

```
rc:/> write_script > foo.g.gz
```

Command Reference for Encounter RTL Compiler

Input and Output

Related Information

Affected by these commands:

- [create_mode](#) on page 287
- [define_clock](#) on page 290
- [define_cost_group](#) on page 295
- [define_dft_scan_chain](#) on page 674
- [external_delay](#) on page 298
- [multi_cycle](#) on page 304
- [path_adjust](#) on page 308
- [path_delay](#) on page 312
- [path_disable](#) on page 315
- [path_group](#) on page 318

Command Reference for Encounter RTL Compiler

Input and Output

write_sdc

```
write_sdc
  [-version {1.1|1.3|1.4|1.5|1.5rc}]
  [-strict] [-no_split] [-exclude cmd_list]
  [-mode mode_name] [design] [> file]
```

Writes out the current design constraints in Synopsys Design Constraint (SDC) format. The `write_sdc` command can also compress the SDC constraints with gzip (.gz extension).

When using the `write_sdc` command, RTL Compiler replaces the / character with the @ character when the / character is used in the name of objects. This could happen when the design is ungrouped or when the / character is used as the `ungroup_separator`.

To prevent this problem, write out the constraints using an SDC version less than 1.3 to avoid the hsc specification. For example: `rc:/> write_sdc -version 1.1.`

For those SDCs that are not supported, RTL Compiler will issue a warning message but store them for output for the `write_sdc` command. RTL Compiler will only store the SDCs and not manipulate any data with them.

Note: Using the `write_sdc` command may not capture all the design information necessary to recreate the image of a design's constraints.

Options and Arguments

<code>design</code>	Specifies the name of the design for which to write the SDC constraints.
<code>file</code>	Specifies the name of the file to which to write the SDC constraints.
<code>-exclude cmd_list</code>	Specifies the commands that must not be written to the SDC file.
<code>-mode mode</code>	Writes out mode specific constraints for a design.
<code>-no_split</code>	Prevents printing the SDC commands over several lines.
<code>-strict</code>	Writes out commands that are specifically listed in the SDC specification. If you do not use this option, the <code>write_sdc</code> command outputs commands that are DC and PT compatible, which means that commands not listed in the SDC specification may be written out. See Examples for the difference in results when using the <code>-strict</code> option.

Command Reference for Encounter RTL Compiler

Input and Output

```
-version {1.1|1.3|1.4|1.5|1.5rc}
```

Specifies the SDC version to use. Version 1.5rc includes the `set_time_unit` and `set_load_unit` commands.

Examples

- The following example writes out the SDC constraints to the `my_des.sdc` file:
`rc:/> write_sdc /designs/my_des > my_des.sdc`
- The following example shows the results you may get if you do not specify the `-strict` option with the `write_sdc` command.

```
#####
...
#####
set sdc_version 1.4
# Set the current design
current_design add

set_wire_load_mode "enclosed"
set_wire_load_selection_group "ALUMINUM" -library "tutorial"
set_dont_touch [get_designs Madd_addinc]
set_dont_touch [get_cells flop1]
```

- The following example shows the results you may get if you do specify the `-strict` option with the `write_sdc` command.

```
#####
...
#####
set sdc_version 1.4

# Set the current design
current_design add

set_wire_load_mode "enclosed"
set_wire_load_selection_group "ALUMINUM" -library "tutorial"
```

- The following example shows how to write out mode-specific constraints:

```
write_sdc -mode mode1 mode1.sdc
```

- The following examples show the effect of the `-no_split` option.

Assume that the `write_sdc` command would write out the following command in the SDC file:

```
set_false_path -from [list \
    [get_clocks in1] \
    [get_clocks in2] ] -to [get_clocks in3]
```

Command Reference for Encounter RTL Compiler

Input and Output

If you specify the `write_sdc` command with the `-no_split` option, the SDC command would be written as:

```
set_false_path -from [list [get_clocks in1] [get_clocks in2] ] -to [get_clocks in3]
```

- The following example illustrates the use of the `-exclude` option.

```
rc:/> write_sdc
# ######
# Created by Encounter(R) RTL Compiler 10.1.200 on Tue Nov 16 12:58:56 -0800 2010
# #####
set sdc_version 1.7

set_units -capacitance 1000.0fF
set_units -time 1000.0ps

# Set the current design
current_design top

set_case_analysis 0 [get_ports in1]
create_clock -name "abc" -add -period 10.0 -waveform {0.0 5.0} [get_ports clk]
create_clock -name "abc1" -add -period 15.0 -waveform {0.0 7.5} [get_ports clk]
set_clock_gating_check -setup 0.0
set_input_delay -clock [get_clocks abc] -add_delay 0.3 [get_ports in]
set_output_delay -clock [get_clocks abc] -add_delay 0.3 [get_ports out]
set_wire_load_mode "enclosed"
set_wire_load_selection_group "ALUMINUM" -library "tutorial"
set_clock_latency 3.1 [get_clocks abc]
set_clock_uncertainty -setup 45.0 [get_clocks abc]
set_clock_uncertainty -hold 45.0 [get_clocks abc]
set_max_time_borrow 3.0 [get_clocks abc]
set_clock_latency 4.0 [get_clocks abc1]

rc:/> write_sdc -exclude "set_clock_latency set_clock_uncertainty \
set_case_analysis create_clock"
or

rc:/> write_sdc -exclude {set_clock_latency set_clock_uncertainty \
set_case_analysis create_clock}

# ######
# Created by Encounter(R) RTL Compiler 10.1.200 on Tue Nov 16 13:05:12 -0800 2010
# #####
set sdc_version 1.7

set_units -capacitance 1000.0fF
set_units -time 1000.0ps

# Set the current design
current_design top

set_clock_gating_check -setup 0.0
set_input_delay -clock [get_clocks abc] -add_delay 0.3 [get_ports in]
set_output_delay -clock [get_clocks abc] -add_delay 0.3 [get_ports out]
```

Command Reference for Encounter RTL Compiler

Input and Output

```
set_wire_load_mode "enclosed"
set_wire_load_selection_group "ALUMINUM" -library "tutorial"
set_max_time_borrow 3.0 [get_clocks abc]
```

Related Information

[Performing Multi-Mode Timing Analysis in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

Affected by this command: [create_mode](#) on page 287

[read_sdc](#) on page 209

Command Reference for Encounter RTL Compiler

Input and Output

write_sdf

```
write_sdf [-version {OVI 3.0 | OVI 2.1}]
           [-precision non_negative_integer]
           [-timescale {ps | ns}] [-delimiter character]
           [-celltiming {all | none | nochecks}]
           [-interconn {port | interconnect [-no_empty_cells]}]
           [-edges {edged | check_edge}] [-condelse]
           [-nonegchecks] [-no_escape] [-nosplit_timing_check]
           [-no_input_port_nets] [-no_output_port_nets]
           [-recrem {merge_always | merge_when_paired | split}]
           [-setup {merge_always | merge_when_paired | split}]
           [-design] [> file]
```

The command generates a Standard Delay Format (SDF) file that analysis and verification tools or timing simulation tools can use for delay annotation. The SDF file specifies the delay of all the cells and interconnects in the design in the Standard Delay Format. Specifically, it includes the delay values for all the timing arcs of a given cell in the design.

Note: Use the `write_sdf` command after technology mapping (after the `synthesize -to mapped` command).

Options and Arguments

-celltiming {all | none | nochecks}

Specifies which cells delays and timing checks to write out.

`all`—Writes all cell delays and timing checks to the SDF file.

none—Excludes cell delays and timing checks from being written into the SDF file.

`nochecks`—Only excludes the timing checks.

Default: all

-condelse Writes CONDELSE constructs with the default value when a COND construct is written.

-delimiter *character*

Specifies the hierarchy divider character to be used in the SDF file. The valid options are the “/” and “.” characters.

-design Specifies the design name for which the SDF file has to be generated.

Command Reference for Encounter RTL Compiler

Input and Output

-edges { edged | check_edge }

Specifies the edges values.

check_edge—Keeps edge specifiers on timing check arcs but does not add edge specifiers on combinational arcs.

edged—Keeps edge specifiers on timing check arcs as well as combinational arcs.

Default: edged

file

Specify the SDF file name.

-interconn { port | interconnect }

Specifies the construct to use for writing out net delays.

port—Writes out the net delays using the PORT construct.

interconnect—Writes out the net delays using the INTERCONNECT construct.

Default: port

-no_escape

Writes out object names without escaping the special characters such as “[“ or ”]”.

-nonegchecks

Converts all negative timing check values to 0 . 0.

-no_empty_cells

Suppresses writing out empty cell descriptions.

-no_input_port_nets

Suppresses writing out nets connected to input ports.

-no_output_port_nets

Suppresses writing out nets connected to output ports.

-nosplit_timing_check

Does not split the TIMINGCHECK delays (SETUP/HOLD/RECOVERY/REMOVAL delays). Instead, the maximum delay values are used.

-precision *non_negative_integer*

Specifies the number of digits appearing after the decimal point in the output SDF file.

Command Reference for Encounter RTL Compiler

Input and Output

-recrem [merge_always | merge_when_paired | split]

Determines how RECOVERY and REMOVAL timing checks are written.

merge_always—Always writes combined RECREM checks, even if either of the RECOVERY or REMOVAL check does not exist.

For example, if only a RECOVERY check exists, the tool writes a combined RECREM check in the SDF file as follows:

```
RECREM (value) (value) () ()
```

split—Splits the RECREM checks into a RECOVERY check and a REMOVAL check.

merge_when_paired—Writes combined RECREM checks only when both RECOVERY and REMOVAL checks exist.

For example, if both a RECOVERY and a REMOVAL check exist, the tool writes a combined RECREM check in the SDF file as follows:

```
RECREM (value) (value) (value) (value)
```

However, if only the RECOVERY check exists, the tool only writes the RECOVERY check in the SDF file:

```
RECOVERY (value) (value)
```

Default: merge_always

-setuphold [merge_always | merge_when_paired | split]

Determines how SETUP and HOLD timing checks are written.

merge_always—Always writes combined SETUPHOLD checks, even if either of the SETUP or HOLD check does not exist.

For example, if only a SETUP check exists, the tool writes a combined SETUPHOLD check in the SDF file as follows:

```
SETUPHOLD (value) (value) () ()
```

split—Splits the SETUPHOLD checks into a SETUP check and a HOLD check.

merge_when_paired—Writes combined SETUPHOLD checks only when both SETUP and HOLD checks exist.

Command Reference for Encounter RTL Compiler

Input and Output

For example, if both a **SETUP** and a **HOLD** check exist, the tool writes a combined **SETUPHOLD** check in the SDF file as follows:

```
SETUPHOLD (value) (value) (value) (value)
```

However, if only the **SETUP** check exists, the tool only writes the **SETUP** check in the SDF file:

```
SETUP (value) (value)
```

Default: merge_always

-timescale {ps | ns}

Specifies the timescale setting of the SDF file in either nanoseconds or picoseconds.

Default: ps

-version {OVI 3.0 | OVI 2.1}

Specifies whether to generate SDF version 2.1 or 3.0.

Default: OVI 3.0

Examples

- The following example writes out the SDF file, `areid.sdf`, with the “/” delimiting character:

```
rc:/> synthesize -to_mapped  
rc:/> write_sdf -delimiter "/" > areid.sdf
```

- The following report illustrates two **TIMINGCHECK** examples: the first only writes out the maximum delays while the second writes out all the delays.

```
(TIMINGCHECK  
(HOLD D (posedge CK) (":0.0))  
(SETUP D (posedge CK) (":0.452))  
)  
  
(TIMINGCHECK  
(HOLD (negedge D) (posedge CK) (":0.0))  
(HOLD (posedge D) (posedge CK) (":0.0))  
(SETUP (negedge D) (posedge CK) (":0.452))  
(SETUP (posedge D) (posedge CK) (":0.181))  
)
```

Command Reference for Encounter RTL Compiler

Input and Output

To write out only the maximum TIMINGCHECK delays (the first report above), use the `-nosplit_timing_check` option:

```
rc:/> write_sdf -nosplit_timing_check > areid.sdf
```

write_set_load

```
write_set_load [design] [> file]
```

Generates a set of load values, which were obtained from the physical layout estimator (PLE) or wire-load model, for all the nets in the specified design. This command is useful when performing timing correlation across various synthesis and timing tools. The `write_set_load` command can be used to reproduce PLE based timing in external timing analyzers.

Options and Arguments

design Generates the load values for the specified design.

file Specifies the file to which to write the load values.

Example

- The following example shows that the load values on all the nets is .0103:

```
rc:/> write_set_load
set_load 0.0103 [get_nets inst1/out1[3]]
set_load 0.0103 [get_nets inst1/out1[2]]
set_load 0.0103 [get_nets inst1/out1[1]]
set_load 0.0103 [get_nets inst1/out1[0]]
```

write_spf

See [write_spf](#) in Chapter 9, “Physical.”

write_sv_wrapper

```
write_sv_wrapper
  [-allow_error]
  [-module_suffix string] [-subdesign_suffix string]
  [-update_design_name] [design] [> file]
```

Writes out a wrapper SystemVerilog module for a design. Such a wrapper module can help in a comparative simulation of the output netlist from RTL Compiler and the input RTL design, especially when the design description in the input RTL has complex ports.

When you synthesize a design test that has complex ports in the input RTL description, the `write_sv_wrapper` command writes out a System Verilog description of module `test` which has the same complex ports as the original RTL description of test. The module's test body consists of a single instantiation of another module named `test%s` where *%s* is the suffix specified with the `-subdesign_suffix` option.

Options and Arguments

<code>-allow_error</code>	Prevents that a blocking error code is returned even if an error occurred during wrapper creation.
<code>design</code>	Writes out the wrapper SystemVerilog module for the specified design.
<code>file</code>	Specifies the file to which to write the wrapper SystemVerilog module.
<code>-module_suffix <i>string</i></code>	Specifies the suffix to append to the module name of the wrapper module.
<code>-subdesign_suffix <i>string</i></code>	Specifies the suffix to append to the name of the instantiated module.
<code>-update_design_name</code>	Renames the main design module and uses the value of the <code>-subdesign_suffix</code> option as suffix to the design name.

Command Reference for Encounter RTL Compiler

Input and Output

Example

Consider the following Verilog RTL input in test.v.

```
// test.v
module test(
    input [1:0][1:0] inx,
    output outx);
    assign outx = | (inx[1] & inx[0]); //'Or' of bit-wise 'and' of inx[0] and inx[1]
endmodule
```

Assume the Verilog input file is synthesized through the script test.g:

```
## test.g
read_hdl -sv test.v
set_attr library tutorial.lib
elaborate
synthesize -to_map
write_hdl > test.vc
```

The output below shows that the 2-dimensional input port inx has been broken up during synthesis into two ports \inx[0] and \inx[1], so that the netlist is a structural Verilog netlist. This mismatch between the ports of the module in the output and the ports of the input RTL module can create problems for simulation-based comparison between the input and output design.

```
// test.vc : the file output by RC
module test(\inx[0] , \inx[1] , outx);
    input [1:0] \inx[0] , \inx[1] ;
    output outx;
    wire [1:0] \inx[0] , \inx[1] ;
    wire outx;
    wire n_0, n_1;
    nand2 g26(.A (n_1), .B (n_0), .Y (outx));
    nand2 g27(.A (\inx[0] [1]), .B (\inx[1] [1]), .Y (n_1));
    nand2 g28(.A (\inx[0] [0]), .B (\inx[1] [0]), .Y (n_0));
endmodule
```

To avoid this port mismatch, the script should be altered as shown below:

```
## test.g
set_hdl_sv_module_wrapper 1
read_hdl -sv test.v
set_attr library tutorial.lib
elaborate
synthesize -to_map
write_sv_wrapper -update_design_name -subdesign_suffix _core test > test_wrap.v
write_hdl > test.vc
```

Command Reference for Encounter RTL Compiler

Input and Output

This results in the following output:

```
// File test_wrap.v
module test(
    input logic [1:0][1:0] inx,
    output logic outx);
    test_core u1(
        .outx(outx),
        .\inx[1] (inx[1]),
        .\inx[0] (inx[0]));
endmodule

// File test.vc
module test_core(\inx[0] , \inx[1] , outx);
    input [1:0] \inx[0] , \inx[1] ;
    output outx;
    wire [1:0] \inx[0] , \inx[1] ;
    wire outx;
    wire n_0, n_1;
    nand2 g26(.A (n_1), .B (n_0), .Y (outx));
    nand2 g27(.A (\inx[0] [1]), .B (\inx[1] [1]), .Y (n_1));
    nand2 g28(.A (\inx[0] [0]), .B (\inx[1] [0]), .Y (n_0));
endmodule
```

By combining the two files (`test_wrap.v` and `test.vc`), you get a modified design hierarchy with the benefit that the top module has the same ports as the original RTL definition of the top module.

Command Reference for Encounter RTL Compiler

Input and Output

write_tcf

Refer to [write_tcf](#) in [Chapter 11, “Low Power Synthesis.”](#)

Command Reference for Encounter RTL Compiler

Input and Output

write_template

```
write_template
    [-dft] [-power] [-cpf] [-retime]
    [-physical] [-performance]
    [-area] [-no_sdc] [-n2n] [-yield]
    [-full] [-simple] [-split]
    [-multimode] -outfile string
```

Generates a template script with the commands and attributes needed to run RTL Compiler. Use the command options to include specific commands and attributes in the script.

Options and Arguments

-area	Writes out a template script for area-critical designs.
-cpf	Writes out a template script for the Common Power Format (CPF) based flow and a template CPF file (<code>template.cpf</code>). The template CPF file is read in the template script with the <code>read_cpf</code> command.
-dft	Writes out a template script for the test synthesis (DFT) flow. The template contains commands and attributes needed for basic and advanced DFT features.
-full	Writes out DFT, power, and retiming commands and attributes along with the basic template.
-multimode	Writes out a template script for multi-mode analysis.
-n2n	Writes out the template script for netlist to netlist optimization in RTL Compiler. Use with the <code>-dft</code> and <code>-power</code> options to include the DFT and power attributes and commands.
-no_sdc	Writes out clock delays and input and output delays in the RTL Compiler format using the <code>define_clock</code> and the <code>external_delay</code> commands.
-outfile <i>string</i>	Specifies the name of the file to which the template script is to be written.
-performance	Writes out a template script which enables some advanced optimization algorithms that can improve QoR and that have an impact on the runtime.
-physical	Writes out a template script for the physical flow.

Command Reference for Encounter RTL Compiler

Input and Output

-power	Writes out power attributes and commands.
-retime	Writes out retiming attributes and commands.
-simple	Writes out a simple template script.
	You cannot use this option with the <code>-split</code> , <code>-area</code> , <code>-dft</code> , <code>-cpf</code> , <code>-multimode</code> , <code>-physical</code> , <code>-power</code> , <code>-retime</code> , or <code>-full</code> options.
-split	<p>Writes out a template script with a separate setup file which contains the root attributes and setup variables.</p> <p>If you specified the <code>-dft</code> option with the <code>-split</code> option, an additional file is created which contains the DFT design attributes, test clock and scan chain information.</p> <p>If you specified the <code>-power</code> option with the <code>-split</code> option, an additional file is created which contains the leakage and dynamic power, and clock-gating setup information.</p>
-yield	Writes out a template script for yield.

Examples

- The following example writes out the basic template file with the constraints in SDC format:

```
rc:/> write_template -outfile template.g
```
- The following example writes out the basic template file with the constraints in the RTL Compiler format using the `define_clock` and the `external_delay` commands:

```
rc:/> write_template -no_sdc -outfile template.g
```
- The following example adds both the DFT and power related attributes to the template.g file written out:

```
rc:/> write_template -dft -power -outfile template.g
```
- The following example writes out the template script with the DFT attributes and commands for netlist to netlist optimization:

```
rc:/> write_template -dft -n2n -outfile template.g
```
- The following example writes out the template script `template.g` and the setup file `setup_template.g` that contains all the root attributes and setup variables and includes it in the `template.g` file:

```
rc:/> write_template -split -outfile template.g
```

Command Reference for Encounter RTL Compiler

Input and Output

- The following example writes out the *template.g* template script, a *setup_template.g* setup file, a *dft_template.g* file, and a *power_template.g* file and includes them in the appropriate *template.g* file:
`write_template -split -dft -power -outfile template.g`

- The following example writes out a simple template script with no path and cost groups and without any variables, power, or DFT related attributes:

```
write_template -simple -outfile template.g
```

- The following example writes out a template script for area critical designs:

```
write_template -area -outfile template.g
```

Constraints

- [clock_uncertainty](#) on page 284
- [create_mode](#) on page 287
- [define_clock](#) on page 290
- [define_cost_group](#) on page 295
- [derive_environment](#) on page 296
- [external_delay](#) on page 298
- [generate_constraints](#) on page 302
- [multi_cycle](#) on page 304
- [path_adjust](#) on page 308
- [path_delay](#) on page 312
- [path_disable](#) on page 315
- [path_group](#) on page 318
- [propagate_constraints](#)
- [specify_paths](#) on page 323
- [validate_constraints](#) on page 329

clock_uncertainty

```
clock_uncertainty
  [-fall | -rise]
  [-from_edge string] [-to_edge string]
  [-from clock_list | -fall_from clock_list |
   -rise_from clock_list]
  [-to clock_list | -fall_to clock_list |
   -rise_to clock_list]
  [-hold | -setup]
  [-clock clock_list] uncertainty
  [clock|port|instance|pin]...
```

Specifies the uncertainty on the clock network. You specify either a simple or an inter-clock uncertainty.

- Simple uncertainties are defined directly on a clock, port, pin, or instance. These uncertainty values are stored in attributes.
- The inter-clock uncertainties (defined using options such as `-from`, `-to`, `-rise_from`, `-rise_to`) are modeled as `path_adjust` exceptions. These uncertainties take precedence over the simple uncertainty values.

Options and Arguments

{*clock* | *pin* | *port* | *instance*}

Specifies the clocks, pins, ports and instances to which the specified uncertainty value applies. In case of instances, the uncertainty is applied on the input pins of the instance.

Note: These arguments only apply to simple uncertainties.

-clock *clock_list* Specifies the clock(s) to which the uncertainty value applies.

If this option is not specified, it applies to all clocks propagating to that pin or port.

Note: This option only applies to simple uncertainties.

-fall | **-rise** Specifies to apply the uncertainty to the falling or rising edge of the capture clock pin.

If neither option is specified, the uncertainty value applies to both edges.

Note: These options only apply to inter-clock uncertainties.

Command Reference for Encounter RTL Compiler

Constraints

`-from clock_list | -fall_from clock_list | -rise_from clock_list`

Applies the uncertainty value to the specified list of launching clocks.

Note: These options only apply to inter-clock uncertainties.

`-from_edge {rise | fall | both}`

Specifies the edge type of the launch clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_from` or `-rise_from`.

`-hold | -setup`

Specifies that the clock uncertainty applies to hold or setup checks.

If neither option is specified, the uncertainty value applies to both checks.

Note: These options apply to both types of uncertainties.

`-to clock_list | -fall_to clock_list | -rise_to clock_list`

Applies the uncertainty value to the specified list of capture clocks.

Note: These options only apply to inter-clock uncertainties.

`-to_edge {rise | fall | both}`

Specifies the edge type of the capture clock.

Note: This option only applies to inter-clock uncertainties and cannot be specified with either `-fall_to` or `-rise_to`.

uncertainty

Specifies the uncertainty value for the clock(s). Use a floating value.

Examples

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of all clocks propagating to `reg1/CK`.

```
clock_uncertainty -setup -rise 0.4 [find / -pin reg1/CK]
```

- The following command sets a (simple) setup uncertainty of 0.4 sdc units for all paths ending at `reg1` and captured by the rising transition of `clk1`.

```
clock_uncertainty -setup -rise -clock clk1 0.4 [find / -pin reg1/CK]
```

Command Reference for Encounter RTL Compiler

Constraints

- The following command sets a (inter clock) setup uncertainty of 0.5 sdc units for all paths launched by `clk2` and captured by `clk1`.

```
clock_uncertainty -setup -from clk2 -to clk1 0.5
```

Related Information

Affects this command: [path_adjust](#)

Related command: [dc::set_clock_uncertainty](#)

Sets these attributes:

- [clock_hold_uncertainty](#)
- [clock_setup_uncertainty](#)
- [hold_uncertainty_by_clock](#)
- [setup_uncertainty_by_clock](#)

create_mode

```
create_mode -name mode_list  
[-default] [-design design]
```

Specifies the mode for power and timing analysis and optimization. Use this command after loading and elaborating the design, before reading in an SDC file, and before using any of the following constraints: [define_clock](#), [external_delay](#), [multi_cycle](#), [path_adjust](#), [path_delay](#), [path_disable](#), [path_group](#), and [specify_paths](#).

After creating modes, use the `-mode` option with the `read_sdc` command.

Note: You cannot use this command in a CPF flow.

The command returns the directory path to the `mode` object that it creates. You can find the objects created by the `create_mode` command in:

```
/designs/design/modes
```

Options and Arguments

<code>-default</code>	Designates the specified mode as the default mode. In this case, you can specify only mode with the <code>-name</code> option.
<code>-design design</code>	Specifies the name of the design for which you want to create a mode.
<code>-name mode_list</code>	Specifies the names of the modes to be created. Unless a mode is assigned to be the default mode using the <code>-default</code> option, the first mode specified will be the default.

Examples

- The following example creates multiple modes for one design using one `create_mode` command:

```
rc:/>create_mode -name "mode1 mode2" -design design_name
```

- The following example creates two modes using multiple `create_mode` commands, and declares one of them as the default mode.

```
rc:/> create_mode -name a  
/designs/design_name/modes/a  
rc:/>create_mode -name b -default  
/designs/design_name/modes/b
```

Command Reference for Encounter RTL Compiler

Constraints

- The following example illustrates the recommended flow.

```
read_hdl test.v
elaborate
create_mode -name {a b}
read_sdc -mode a a.sdc
read_sdc -mode b b.sdc
```

Related Information

[Creating Modes in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#) for detailed information.

Affects these commands:	define_clock on page 290 external_delay on page 298 multi_cycle on page 304 path_adjust on page 308 path_delay on page 312 path_disable on page 315 path_group on page 318 read_sdc on page 209 report_clocks on page 397 report_timing on page 498 specify_paths on page 323 write_sdc on page 265
Related commands:	derive_environment on page 296 report_summary on page 492 write_encounter on page 243 write_script on page 262
Sets this attribute:	default
Related attributes:	(instance) disabled_arcs_by_mode (pin/port) external_delays_by_mode (design-instance) latch_borrow_by_mode (design-instance/pin) latch_max_borrow_by_mode

Command Reference for Encounter RTL Compiler

Constraints

(pin/port) propagated clocks by mode

(design/pin/port/cost group) slack by mode

(pin/port) timing case computed value by mode

(instance) timing case disabled arcs by mode

(pin/port) timing case logic value by mode

define_clock

```
define_clock -name string
  [-period integer] [-divide_period integer]
  [-rise integer] [-divide_rise integer]
  [-fall integer] [-divide_fall integer]
  [-domain string] [-mode mode_name]
  [-design design] [pin|port]...
```

Defines a clock waveform. A clock waveform is a periodic signal with one rising edge and one falling edge per period. The command returns the directory path to the clock object that it creates.

Note: Clock waveforms that are not applied to objects in your design are referred to as “external” clocks and are only used as references for external delay values (see the [external_delay](#) command).

Options and Arguments

-design *design* Specifies the name of the top module for which you want to define a clock waveform.

This option is required for external clocks when there are multiple top designs or user netlists.

-divide_fall *integer*

Determines together with the **-fall** option the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing **-fall** by **-divide_fall**.

Default: 100

-divide_period *integer*

Determines together with the **-period** option the clock period interval. The clock period is specified in picoseconds and is derived by dividing **-period** by **-divide_period**.

Default: 1

Command Reference for Encounter RTL Compiler

Constraints

`-divide_rise integer`

Determines, together with the `-rise` option, the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 100

`-domain string`

Specifies the name of the clock domain. A clock domain groups clocks that are synchronously related to each other, allowing timing analysis to be performed between these clocks. RTL Compiler only computes timing constraints between clocks in the same clock domain.

Paths between clocks in different domains are unconstrained by default. To constrain these paths, use the [path_delay](#) command.

Default: domain_1

`-fall integer`

Determines, together with the `-divide_fall` option, the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 50 (falling edge is halfway through the period)

`-mode mode_name`

Defines a clock waveform for a mode.

`-name string`

Specifies the name of the clock that is being defined.

Each clock object in your design must have a unique name. If you define a new clock with the same name as an existing clock, then the new clock replaces the old one.

The clock name allows you to search for the clock later (through the [find](#) command) or to recognize it in reports.

Default: Name of the first pin or port object specified

`-period integer`

Determines, together with the `-divide_period` option, the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

`{pin | port}`

Specifies the clock input pin or port.

You can apply clock waveforms to input ports of your design, hierarchical pins, clock pins of sequential cells in your design, a combination, or to no objects at all.

-rise integer

Determines together with the **-divide_rise** option the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a fraction of the period and is derived by dividing **-rise** by **-divide_rise**.

Default: 0 (rising edge is at the start of the period)

Examples

- The following example defines a clock for a design with top module alu.

```
rc:/> define_clock -period 10000 -name 100MHz -design /designs/alu
```

The clock period is 10,000 picoseconds.

- The following example defines a 300 MHz clock that applies to all sequential logic within a design:

```
rc:/> define_clock -period 10000 -name 300MHz -divide_period 3 [clock_ports]
```

The clock period is 10,000/3 picoseconds. This allows RTL Compiler to compute that there are exactly 3 periods of clock 300MHz to every period of 100MHz.

If you had specified a period of 3,333 picoseconds for the 300 MHz clock, RTL Compiler would compute a different relationship between the clocks (3333 periods of one clock to 10000 periods of the other) and the timing analysis of the design would be different.

- The following example defines clock 100MHz with a rising edge after 20 percent of the period and a falling edge after 80 percent:

```
rc:/> define_clock -period 10000 -name 100MHz -rise 20 -fall 80
```

- The following example defines clock 100MHz with the falling edge 1/3 and the rising edge 2/3 of the way through the period:

```
rc:/> define_clock -period 10000 -name 100MHz -rise 2 -divide_rise 3 \
==> -fall 1 -divide_fall 3
```

Note: The **-divide_rise** and **-divide_fall** options allow you to precisely define when the clock transitions occur. In some cases RTL Compiler needs this precise definition to compute the correct timing constraints for paths that are launched by one clock and captured by another.

Command Reference for Encounter RTL Compiler

Constraints

- The following examples create a clock domain `system` and assign the original 100 MHz and 300 MHz clocks to it:

```
rc:/> define_clock -domain "system" -period 10000 -name 100MHz
rc:/> define_clock -domain "system" -period 10000 -name 300MHz \
==> -divide_period 3 [clock_ports]
```

- The following example saves the directory path to the clock that is defined in variable `clock1`:

```
rc:/> set clock1 [define_clock -period 10000 -name 100MHz]
```

Alternatively, the `find` command can be used to perform a search at a later time.

- The following example removes the clock whose definition you saved in variable `clock1`:

```
rc:/> rm $clock1
```

Note: When you remove a clock object, any external delays that reference it are also removed. Timing exceptions referring to the clock object are also removed if they can't be satisfied without the clock.

- The following example searches for a clock object by name:

```
rc:/> find / -clock 100MHz
```

- The following example examines the attributes of a clock object:

```
rc:/> ls -a [find / -clock 100MHz]
```

Note: The clock object is identified by its directory path.

Related Information

[Defining the Clock Period in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#).

Affects these commands:

[clock_ports](#) on page 372
[external_delay](#) on page 298
[multi_cycle](#) on page 304
[path_adjust](#) on page 308
[path_delay](#) on page 312
[path_disable](#) on page 315
[path_group](#) on page 318
[report_clocks](#) on page 397
[report_qor](#) on page 477

Command Reference for Encounter RTL Compiler

Constraints

[specify_paths](#) on page 323
[read_sdc](#) on page 209
[report_clocks](#) on page 397
[report_summary](#) on page 492
[report_timing](#) on page 498
[synthesize](#) on page 348
[write_encounter](#) on page 243
[write_script](#) on page 262
[write_sdc](#) on page 265

Related command: [create_mode](#) on page 287

Affects these attributes:

[divide_fall](#)
[divide_period](#)
[divide_rise](#)
[fall](#)
[period](#)
[rise](#)
[propagated_clocks_by_mode](#)

define_cost_group

```
define_cost_group -name string
    [-weight integer] [-design design]
```

Defines a cost group. The command returns the directory path to the object that it creates.

Options and Arguments

<code>-design <i>design</i></code>	Specifies the name of the design for which you want to define the cost group.
<code>-name <i>string</i></code>	Specifies the name of the cost group. <i>Default:</i> grp_x
<code>-weight <i>integer</i></code>	Specifies the weight of the cost group. The weight factor is only taken into account during incremental optimization. When two paths have identical cost factors, the tool will pick the one with the highest weight. <i>Default:</i> 1

Examples

The following example assigns a weight factor of 1 to cost group I20 for the top-level design.

```
rc:/> define_cost_group -name I20 -weight 1
/designs/.../timing/exceptions/path_groups/I20
```

Related Information

[Creating Path Groups and Cost Groups](#) in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

[Generating Timing Reports](#) in *Using Encounter RTL Compiler*

Affects these commands:	path_group on page 318 report_timing on page 498 write_script on page 262 write_script on page 262
Sets this attribute:	weight

derive_environment

```
derive_environment [-name string] [-sdc_only] instance
```

Creates a new design from the specified instance and creates timing constraints for all modes for this design based on the timing constraints that the instance had in the original design.

This command does not perform time-budgeting. The slack at each pin in the new design matches the slack at the corresponding pin in the original design.

Note: By default, the `derive_environment` command uses RTL Compiler's more powerful constraint language that produces a more accurate timing view, but is not understood by other tools. Use the `-sdc_only` option to specify that RTL Compiler only apply constraints to the new design that can be expressed in SDC.

If you use the `derive_environment` command to generate constraints for a subdesign then try to write them out, often the constraints cannot be expressed in SDC and you will get the following error message:

```
Error   : The design contains constraints which have no SDC equivalent.[SDC-19]
         : The design is /designs/Madd_addinc.
         : If the design constraints were created using the 'derive_environment'
           command, use the '-sdc_only' option so that only constraints that can be expressed
           in SDC are generated. By default the 'derive_environment' command uses the more
           powerful RC constraints, which cannot always be converted to SDC.
```

Options and Arguments

<i>instance</i>	Specifies the name of the instance whose environment (constraints) you want to derive.
<code>-name <i>string</i></code>	Specifies the name of the target design.
<code>-sdc_only</code>	Specifies that the tool only apply constraints to the new design that can be expressed in SDC. Using this option makes the new constraints less accurate, but makes it possible to use the constraints in flows between different tools that support SDC. By default, the <code>derive_environment</code> command uses RTL Compiler's more powerful constraint language that produces a more accurate timing view, but is not understood by other tools.

Command Reference for Encounter RTL Compiler

Constraints

Related Information

[Synthesizing Submodules in Using Encounter RTL Compiler](#)

[Writing out the Multi-Mode Environment in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

Affected by this command:	<u>path_adjust</u> on page 308 <u>create_mode</u> on page 287
Affects these commands:	<u>report_timing</u> on page 498 <u>synthesize</u> on page 348
Sets this attribute:	<u>precluded_path_adjusts</u>

external_delay

```
external_delay
  {-input min_rise min_fall max_rise max_fall
   |-output min_rise min_fall max_rise max_fall}
   [-clock object] [-edge_fall | -edge_rise]
   [-level_sensitive] [-accumulate]
   [-mode mode_name] [-name string] {port|pin}...
```

Constrains ports and pins within your design. Timing is specified as either an input or output delay, and is specified relative to a clock edge.

External delays are most often specified on top-level ports of your design.

Options and Arguments

-accumulate Indicates that more than one external delay can be specified per clock per phase.

-clock object Specifies the reference clock. Input and output delays are defined relative to this clock.

For an input delay, the reference clock is called the *launching* clock.

For an output delay, the reference clock is called the *capturing* clock.

The reference clock must have been defined with the [define_clock](#) command.

[-edge_rise | -edge_fall]

Specifies to use the rising or falling edge of the reference clock as reference edge.

Default: -edge_rise

-input min_rise min_fall max_rise max_fall

Specifies an input delay. That is the time between the reference edge of the launching clock and the time when the input signal at the specified ports or pins becomes stable.

Command Reference for Encounter RTL Compiler

Constraints

	<p>You can specify one, two, or four integers. If you specify one value, that value is used for all four delay values. If you specify two values, the first value defines the (minimum and maximum) rise delays, while the second value defines the (minimum and maximum) fall delays.</p> <p>The <i>minimum</i> rise (fall) delays are used for hold analysis. The <i>maximum</i> rise (fall) delays are used for setup analysis.</p> <p>Note: RTL Compiler does not support hold analysis.</p> <p>The (minimum and maximum) <i>rise</i> delays are measured with respect to the rising edge of the input signal.</p> <p>The (minimum and maximum) <i>fall</i> delays are measured with respect to the falling edge of the input signal.</p>
-level_sensitive	Used with the <code>-input</code> option, it specifies the constraint coming from a level-sensitive latch.
-mode mode_name	Used with the <code>-output</code> option, it specifies the constraint to a level-sensitive latch.
-name string	Constrains ports and pins by mode in a design.
	Associates a name with the specified timing constraint. If another timing constraint already exists with that name, the existing timing constraint is replaced with the new one.
	The name may be useful for later finding the constraint (with the <code>find</code> command) and for recognizing the constraint in reports.
	<p><i>Default:</i> <code>xx_n</code></p>
-output min_rise min_fall max_rise max_fall	<p>Specifies an external output delay. That is the delay between the time when the output signal at the specified ports or pins becomes stable and the reference edge of the capturing clock.</p> <p>You can specify one, two, or four integers. If you specify one value, that value is used for all four delay values. If you specify two values, the first value defines the (minimum and maximum) rise delays, while the second value defines the fall delays.</p> <p>The <i>minimum</i> rise (fall) delays are used for hold analysis. The <i>maximum</i> rise (fall) delays are used for setup analysis.</p> <p>Note: RTL Compiler does not support hold analysis.</p>

Command Reference for Encounter RTL Compiler

Constraints

The (minimum and maximum) *rise* delays are measured with respect to the rising edge of the output signal.

The (minimum and maximum) *fall* delays are measured with respect to the falling edge of the output signal.

{ <i>pin</i> <i>port</i> }	Specifies delay for timing start points and end points, which can be primary ports, pins of sequential instances, and pins of unresolved references. Use the break_timing_paths attribute to make a non-start/end point a start/end point, which then can be used with the external_delay command.
------------------------------	---

Examples

- The following example specifies an input delay of 300 picoseconds on all bits of port *a* relative to the falling edge of clock *clock1*:

```
rc:/> external_delay -input 300 -edge_fall -clock [find / -clock clock1] \
[find / -port a*]
```

RTL Compiler interprets this as a worst-case upper bound constraint, that is, the latest time to set up the data on a D pin of an edge-triggered flop.

Applying delays to individual bits of multibit ports or busses is possible since each bit of a port is accessible individually within the directory structure of the design.

- The following example specifies an output delay of 1300 picoseconds on all bits of port *a* relative to the rising edge (default) of clock *clock1*:

```
rc:/> external_delay -output 1300 -clock [find / -clock clock1] \
[find / -port a*]
```

RTL Compiler interprets this as a minimum setup time for external logic.

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

- [Setting Input Delays](#)
- [Setting Output Delays](#)
- [Performing Multi-Mode Timing Analysis](#)

Affected by this command: [define_clock](#) on page 290

Command Reference for Encounter RTL Compiler

Constraints

Affects these commands:	report qor on page 477 report timing on page 498 synthesize on page 348 write encounter on page 243 write sdc on page 265 write script on page 262
Related commands:	create mode on page 287 define clock on page 290 derive environment on page 296 multi cycle on page 304 path adjust on page 308 path delay on page 312 path disable on page 315 path group on page 318 specify paths on page 323
Affects these attributes:	External Delay Attributes
Related attributes:	(pin/port) external delays by mode

generate_constraints

```
generate_constraints [-rtl] [-netlist string]
                     [-slack integer] [-report string]
                     [-in_sdc string] [-out_sdc string]
                     {-fpfgen | -dfpfgen | -trv} [> file]
```

Verifies the false paths in the SDC files against the RTL or netlist and then generates any missing functional false paths. The command can be used any time after elaboration. If you do not specify either the `-rtl`, `-netlist`, or `-in_sdc` options, RTL Compiler will internally generate the SDC constraints and verify them against the design at its current state. RTL Compiler will generate any missing constraints.

Use this command in the False Path Generation, Directed False Path Generation, and Timing Report Validation flows.

Options and Arguments

<code>-dfpfgen</code>	Generates the false paths from Directed False Path Generation flow.
<code>file</code>	Specifies the name of the file to write the report.
<code>-fpfgen</code>	Generates the false paths from False Path Generation flow.
<code>-in_sdc <i>string</i></code>	Specifies the UNIX path to the SDC files.
<code>-logfile <i>string</i></code>	Creates a separate CCD logfile. You must specify the UNIX path to the file.
<code>-netlist <i>string</i></code>	Specifies the UNIX path to the netlist.
<code>-out_sdc <i>string</i></code>	Specifies the name for the RTL Compiler generated SDC file. <i>Default:</i> cfp.sdc
<code>-report <i>string</i></code>	Specifies the name of the timing report file to be generated for the design. The file will be in the CCD format.
<code>-rtl</code>	Indicates that the verification should happen against the RTL.
<code>-slack <i>integer</i></code>	Specifies a slack value in picoseconds. Only paths below this slack value will be used for generating the timing report. This must be used with the <code>-report</code> option.
<code>-trv</code>	Generates the false paths from the Timing Report Validation flow.

Command Reference for Encounter RTL Compiler

Constraints

Examples

- The following command generates the false path from the False Path Generation flow, verifies against the RTL, specifies the input SDC file `top.sdc`, and specifies the output SDC file `top_out.sdc`:

```
rc:/> generate_constraints -fpgen -rtl -in_sdc /home/test/top.sdc \
    -out_sdc /home/cody/top_out.sdc
```

- The following command generates the false path from the Directed False Path Generation flow, verifies against the netlist `generic.nl.v`, specifies the input SDC file `top.sdc`, specifies the slack (2 picoseconds), specifies the report name, and specifies the output SDC file `top_out.sdc`:

```
rc:/> generate_constraints -dfpgen -netlist generic.nl.v -in_sdc top.sdc \
    -report top_out.rep -slack 2 -out_sdc top_out.sdc
```

Related Information

[Using the Generate Flow without Dofiles in Interfacing between Encounter RTL Compiler and Encounter Conformal](#)

Affected by this attribute: [wccd_threshold_percentage](#)

Related command: [validate_constraints](#) on page 329

multi_cycle

```
multi_cycle
{ {-from {instance|external_delay|clock|port|pin}...
  |-through {instance|port|pin}...[-through...]...
  |-to {instance|external_delay|clock|port|pin}...}...
  |-paths string}
[-launch_shift integer] [-capture_shift integer]
[-setup] [-hold]
[-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that overrides the default clock edge relationship for paths that meet the path selection criteria. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options. The command returns the directory path to the object that it creates.

RTL Compiler normally computes timing constraints for paths based on the launching clock waveforms and capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

Options and Arguments

`-capture_shift integer`

Specifies the capture clock shift value.

An ordinary two-cycle path would be specified using `-capture_shift 2`. Incrementing the `-capture_shift` value adds a cycle to the path by shifting the capture edge one period later.

Default: 1

`-hold`

Specifies that the exception is for hold timing analysis only.

Default: setup

`-from {instance|external_delay|clock|port|pin}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

Command Reference for Encounter RTL Compiler

Constraints

`-launch_shift integer`

Specifies the launch clock shift value. Incrementing the `-launch_shift` value adds a cycle to the path by shifting the launch edge one period earlier.

Adjusting the launch edge is only useful if the launch and capture clocks have different periods. Otherwise an equivalent timing relationship can be achieved by shifting the capture clock instead.

Default: 0

`-lenient`

Converts an invalid start or end point into a through point and issues a warning message. Without this option, an invalid start or end point results in an error message.

`-mode mode_name`

Creates a timing exception object for a mode that overrides the default clock edge relationship for paths that meet the path selection criteria.

`-name string`

Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one.

The name can be useful for later finding the exception (with the [find command](#)) and for recognizing the exception in reports.

Default: mc_n

`-paths string`

Specifies the paths to which the exception should be applied. The string argument should be created using the [specify_paths command](#). The `-paths` option cannot be used in conjunction with any of the other three path selection options (`-from`, `-through`, `-to`).

`-setup`

Specifies that the exception is for setup timing analysis only.

Default: setup

`-through {instance|port|pin}`

Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

Command Reference for Encounter RTL Compiler

Constraints

-to {instance | external_delay | clock | port | pin}

Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example requires a path to first pass through object a, then end on object b:
`rc:/> multi_cycle -through a -to b`
- The following example requires a path to pass through either object a or b.
`rc:/> multi_cycle -through {a b}`
- The following example requires a path to first pass through object a or b, then pass through object c or d.
`multi_cycle -through {a b} -through {c d}`

The candidate paths would be:

ac ad bc bd

- The following example requires a path that goes through object a, b, and c in that order:
`rc:/> multi_cycle -through a -through b -through c`
- The following example uses a Tcl variable to save the timing exception for future reference:

`rc:/> set two_cycle [multi_cycle -capture_shift 2 -from [find / -port a]]`

The following example removes the timing exception that you saved in the `two_cycle` variable:

`rc:/> rm $two_cycle`

- The following example searches for a timing exception that you defined earlier:
`rc:/> multi_cycle -capture_shift 2 -from [find / -port a] -name two_cycle
/designs/alu/timing/exceptions/multi_cycles/two_cycle`
...
...
`rc:/> find / -exception two_cycle
/designs/alu/timing/exceptions/multi_cycles/two_cycle`
- The following example lists multi cycle timing exception objects using the `ls` command.
`rc:/> ls -l timing/exceptions/multi_cycles`

Command Reference for Encounter RTL Compiler

Constraints

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
multi_cycle -from clk1  
multi_cycle -paths [specify_paths -from clk1]
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

- [Setting Timing Exceptions](#)
- [Overriding the Default Timing Constraint for Multi-Cycle Paths](#)
- [Performing Multi-Mode Timing Analysis](#)

Affects these commands:

[report qor](#) on page 477

[report timing](#) on page 498

[specify_paths](#) on page 323

[synthesize](#) on page 348

[write_encounter](#) on page 243

[write_sdc](#) on page 265

[write_script](#) on page 262

Related commands:

[create_mode](#) on page 287

[define_clock](#) on page 290

[external_delay](#) on page 298

[path_adjust](#) on page 308

[path_delay](#) on page 312

[path_disable](#) on page 315

[path_group](#) on page 318

[specify_paths](#) on page 323

Sets these attributes:

[Exception Attributes](#)

path_adjust

```
path_adjust -delay integer
  { {-from {instance|external_delay|clock|port|pin}...
    |-through {instance|port|pin}...[-through...]...
    |-to {instance|external_delay|clock|port|pin}...}...
    |-paths string }
  [-lenient] [-mode mode_name] [-name string]
```

Modifies path constraints. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

These path constraints could have been

- Computed by the timing engine using the launching and capturing waveforms
- Set explicitly with the `path_delay` command

The command returns the directory path to the object that it creates.

The constraints specified with the `path_adjust` command can co-exist with other timing exceptions, such as `path_delay`, `multi_cycle`, as well as `path_adjust` (it is possible to have multiple `path_adjust` constraints on a path).

The constraints created by the `path_adjust` commands can be found in:

`/designs/..../timing/exceptions/path_adjusts`

Options and Arguments

`-delay integer` Specifies the delay constraint value, in picoseconds, by which the path has to be adjusted. A positive adjustment relaxes the clock constraint and a negative adjustment tightens it.

`-from {instance|external_delay|clock|port|pin}` Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

Command Reference for Encounter RTL Compiler

Constraints

-lenient	Converts an invalid start or end point into a through point and issues a warning message. Without this option, an invalid start or end point results in an error message.
-mode <i>mode_name</i>	Modifies path constraints for a specified mode.
-name <i>string</i>	Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one. The name may be useful for later finding the exception (with the <u>find</u> command) and for recognizing the exception in reports.
	<i>Default:</i> adj_n
-paths <i>string</i>	Specifies the paths to which the exception should be applied. The string argument should be created using the <u>specify_paths</u> command. The -paths option cannot be used in conjunction with any of the other three path selection options (-from, -through, -to).
-through { <i>instance</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances. You can repeat the -through option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.
-to { <i>instance</i> <i>external_delay</i> <i>clock</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies. Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example removes the timing exception that you saved in the `override` variable:

```
rc:/> set override [path_adjust -to $clock -delay -500]  
rc:/> rm $override
```

- The following example searches for a timing exception that you defined earlier:

```
rc:/> path_adjust -to $clock -delay -500 -name override  
/designs/alu/timing/exceptions/path_adjusts/override  
...  
rc:/> find / -exception override  
/designs/alu/timing/exceptions/path_adjusts/override
```

- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
path_adjust -from clk1  
path_adjust -paths [specify_paths -from clk1]
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

- [Creating Path Groups and Cost Groups](#)
- [Modifying Path Constraints](#)
- [Generating Post-Synthesis Timing Reports](#)
- [Performing Multi-Mode Timing Analysis](#)

Affects these commands:

[report timing](#) on page 498

[report qor](#) on page 477

[report timing](#) on page 498

[specify_paths](#) on page 323

[synthesize](#) on page 348

[write_script](#) on page 262

[write_encounter](#) on page 243

[write_sdc](#) on page 265

[write_script](#) on page 262

Command Reference for Encounter RTL Compiler

Constraints

Related commands	define_clock on page 290 external_delay on page 298 multi_cycle on page 304 path_delay on page 312 path_disable on page 315 path_group on page 318 specify_paths on page 323
Sets these attributes:	Exception Attributes

path_delay

```
path_delay -delay integer
  {-from {instance|external_delay|clock|port|pin}...  

   |-through {instance|port|pin}...[-through...]...  

   |-to {instance|external_delay|clock|port|pin}...}  

  |-paths string  

  [-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that allows you to specify the timing constraint for paths that meet the path selection criteria. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options. The command returns the directory path to the object that it creates.

RTL Compiler normally computes timing constraints for paths based on the launching clock waveforms and capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

The constraints created by the `path_delay` commands can be found in:

```
/designs/..../timing/exceptions/path_delays
```

Options and Arguments

<code>-delay <i>integer</i></code>	Specifies the delay constraint value, in picoseconds, for paths that meet the path selection criteria.
<code>-from {<i>instance external_delay clock port pin</i>}</code>	Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies. Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.
<code>-lenient</code>	Converts an invalid start or end point into a through point and issues a warning message. Without this option, an invalid start or end point results in an error message.
<code>-mode <i>mode_name</i></code>	Creates a timing exception object for the specified mode that lets you specify the timing constraint for paths that meet the path selection criteria.

Command Reference for Encounter RTL Compiler

Constraints

-name <i>string</i>	Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one. The name may be useful for later finding the exception (with the <u>find</u> command) and for recognizing the exception in reports. <i>Default:</i> del_n
-paths <i>string</i>	Specifies the paths to which the exception should be applied. The string argument should be created using the <u>specify_paths</u> command. The -paths option cannot be used in conjunction with any of the other three path selection options (-from, -through, -to).
-through { <i>instance</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances. You can repeat the -through option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.
-to { <i>instance</i> <i>external_delay</i> <i>clock</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies. Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example specifies a path delay of 5000 ps for all paths starting from port a .

```
rc:/> path_delay -delay 5000 -from [find / -port a]
```
- The following command defines path delay my_delay of 4000ps for all paths ending at port b .

```
rc:/> path_delay -delay 4000 -to [find / -port b] -name my_delay  
/designs/alu/timing/exceptions/path_delays/my_delay
```

Command Reference for Encounter RTL Compiler

Constraints

The following command searches for the timing exception `my_delay` defined earlier:

```
rc:/> find / -exception my_delay  
/designs/alu/timing/exceptions/path_delays/my_delay
```

- The following examples are equivalent and define a path delay for all paths starting from (clock) pin `clk1`:

```
path_delay -from clk1  
path_delay -paths [specify_paths -from clk1]
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*:

- [Creating Clock Domains](#)
- [Setting Timing Exceptions](#)
- [Modifying Path Constraints](#)
- [Performing Multi-Mode Timing Analysis](#)

Affects these commands:	report qor on page 477 report timing on page 498 specify_paths on page 323 synthesize on page 348 write_script on page 262 write_sdc on page 265
-------------------------	---

Related commands:	create_mode on page 287 define_clock on page 290 external_delay on page 298 multi_cycle on page 304 path_adjust on page 308 path_disable on page 315 path_group on page 318 specify_paths on page 323
-------------------	--

Sets these attributes:	Exception Attributes
------------------------	--------------------------------------

path_disable

```
path_disable
  {-from {instance|external_delay|clock|port|pin}...
   |-through {instance|port|pin}...[-through...]...
   |-to {instance|external_delay|clock|port|pin}...}...
   |-paths string}
  [-lenient] [-mode mode_name] [-name string]
```

Creates a timing exception object that allows you to unconstrain paths. The command returns the directory path to the object that it creates. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

RTL Compiler computes timing constraints for paths based on the launching clock waveforms and the capturing clock waveforms. The default timing constraint is the smallest positive difference that exists between a launching clock edge and a capturing clock edge.

Options and Arguments

`-from {instance | external_delay | clock | port | pin}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which the specified external delay timing exception applies.

Only paths that start at one of the ports or pins, or paths that are launched by one of the clock objects will have the timing exception applied to them.

`-lenient`

Converts an invalid start or end point into a through point and issues a warning message. Without this option, an invalid start or end point results in an error message.

`-mode mode_name`

Creates a timing exception object for the specified mode that lets you unconstrain paths.

`-name string`

Associates a name with the specified timing exception. If another timing exception already exists with that name, the existing timing exception is replaced with the new one.

The name may be useful for later finding the exception (with the `find` command) and for recognizing the exception in reports.

Default: dis_n

Command Reference for Encounter RTL Compiler

Constraints

-paths <i>string</i>	Specifies the paths to which the exception should be applied. The string argument should be created using the <code>specify_paths</code> command. The <code>-paths</code> option cannot be used in conjunction with any of the other three path selection options (<code>-from</code> , <code>-through</code> , <code>-to</code>).
-through { <i>instance</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances. You can repeat the <code>-through</code> option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.
-to { <i>instance</i> <i>external_delay</i> <i>clock</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies. Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects, have the exception applied to them.

Examples

- The following example uses a Tcl variable to save the timing exception for future reference:

```
rc:/> set false_path [path_disable -from [find / -port a]]
```
- The following example removes the timing exception that you saved in variable `false_path`:

```
rc:/> rm $false_path
```
- The following example lists timing exception objects using the `ls` command.

```
rc:/> ls -l timing/exceptions/path_disables
```
- The following examples are equivalent and apply to paths starting from (clock) pin `clk1`:

```
path_disable -from clk1
path_disable -paths [specify_paths -from clk1]
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*:

- [Specifying a False Path](#)
- [Performing Multi-Mode Timing Analysis](#)

Affects these commands:	<u>report qor</u> on page 477 <u>report timing</u> on page 498 <u>specify_paths</u> on page 323 <u>synthesize</u> on page 348 <u>write_encounter</u> on page 243 <u>write_script</u> on page 262 <u>write_sdc</u> on page 265
Affected by these commands:	<u>define_clock</u> on page 290 <u>multi_cycle</u> on page 304
Related commands:	<u>create_mode</u> on page 287 <u>define_clock</u> on page 290 <u>external_delay</u> on page 298 <u>path_adjust</u> on page 308 <u>path_delay</u> on page 312 <u>path_group</u> on page 318 <u>specify_paths</u> on page 323
Sets these attributes:	<u>Exception Attributes</u>

path_group

```
path_group
  {-from {instance|external_delay|clock|port|pin}...
   |-through {instance|port|pin}...[-through...]...
   |-to {instance|external_delay|clock|port|pin}...}...
   |-paths string}
  [-group string]
  [-lenient] [-mode mode_name] [-name string]
```

Assigns paths that meet the path selection criteria to a cost group. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these four options. The `-paths` option cannot be used in conjunction with any of the other three path selection options.

Options and Arguments

<code>-from {instance external_delay clock port pin}</code>	Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports.
<code>-group string</code>	Specifies the name of the cost group (defined with <code>define_cost_group</code>) to which the path is added.
<code>-lenient</code>	Converts an invalid start or end point into a through point and issues a warning message. Without this option, an invalid start or end point results in an error message.
<code>-mode mode_name</code>	Assigns paths for a specified mode that meet the path selection criteria to a cost group.
<code>-name string</code>	Associates a name with the specified timing exception.
<code>-paths string</code>	Specifies the paths to which the exception should be applied. The string argument should be created using the <code>specify_paths</code> command. The <code>-paths</code> option cannot be used in conjunction with any of the other three path selection options (<code>-from</code> , <code>-through</code> , <code>-to</code>).
<code>-through {instance port pin}</code>	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances.

Command Reference for Encounter RTL Compiler

Constraints

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {instance | external_delay | clock | port | pin}`

Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports.

Examples

- The following example assigns the paths from all inputs to all outputs to the group called I2O, which was previously defined by a `define_cost_group` command:

```
path_group -from /designs/*/ports_in/* -to /designs/*/ports_out/* -group I2O
```

- The following examples are equivalent and apply to paths starting from (clock) pin clk1:

```
path_group -from clk1  
path_group -paths [specify_paths -from clk1]
```

Related Information

[Creating Path Groups and Cost Groups](#) in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*.

[Performing Multi-Mode Timing Analysis](#) in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affects these commands:

[define_cost_group](#) on page 295

[report qor](#) on page 477

[report timing](#) on page 498

[specify_paths](#) on page 323

[synthesize](#) on page 348

[write_encounter](#) on page 243

[write_script](#) on page 262

[write_sdc](#) on page 265

Command Reference for Encounter RTL Compiler

Constraints

Related commands:	create mode on page 287 define clock on page 290 external delay on page 298 multi cycle on page 304 path adjust on page 308 path delay on page 312 path disable on page 315 specify paths on page 323
Sets these attributes:	Exception Attributes

propagate_constraints

```
propagate_constraints
  -block_sdc string
  [-glue_sdc string]
  [-partial_chip_sdc string]
  [-out_sdc string] [-netlist string]
  [-rule_instance_file string]
  [-rule_instance_template string]
  [-logfile string] [> file]
```

Propagates the block-level constraints to the top-level and integrates them to generate a chip-level constraints. Propagating and integrating of the design constraints requires to analyze the provided block-level and glue constraints, and to check them against any existing chip-level constraints to determine whether to ignore or promote them to the top-level.

Options and Arguments

<i>-block_sdc string</i>	Specifies a list of block names with their associated block-level SDC files in the following format: {{block_name <i>block_sdc_file</i> }...}
Note: You need to specify the path to the SDC files. If the paths contain Tcl variables, use the format shown in the Examples on page 322.	
<i>file</i>	Specifies the file to which the report must be written.
<i>-glue_sdc string</i>	Specifies the name of a glue SDC file. This file contains a set of constraints for the top-level module only (without covering any block-level constraints).
<i>-logfile string</i>	Creates a separate CCD logfile. You must specify the UNIX path to the file.
<i>-netlist string</i>	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
<i>-out_sdc string</i>	Specifies the name of the constraints file that is generated after propagation and integration of the block and glue constraints. <i>Default:</i> chip.sdc
<i>-partial_chip_sdc string</i>	Specifies the name of the partial SDC file that corresponds to the top-level of the design.

Command Reference for Encounter RTL Compiler

Constraints

-rule_instance_file string

Specifies the name of the file which defines the rule instances for the Encounter® Conformal® Constraint Designer (CCD) tool.

-rule_instance_template string

Creates a template with the default rules. You can modify this file according to the design. To use this file as input for the Encounter® Conformal® Constraint Designer (CCD) tool, specify the file as value of the **-rule_instance_file** option.

Examples

- The following command creates a top-level SDC file, `chip.sdc`, which integrates the block-level SDC files `i1.sdc` and `i2.sdc`.

```
rc:/> propagate_constraints -block_sdc {{i1 i1.sdc} {i2 i2.sdc}}
```

The internally generated CCD dofile will be similar to:

```
read library -statetable -liberty ./tech/slow.lib
add search path -design .
read design -verilog ./ti1.v -lastmod -noelab
elaborate design
dofile ./default_rule_instances.do
read hierarchical sdc \
-sdc_design i1 i1.sdc \
-sdc_design i2 i2.sdc
set system mode verify
integrate -all ./chip.sdc -replace
report rule check
report environment
```

- The following example shows the use of Tcl variables to specify the paths to the SDC files:

```
propagate_constraints -block_sdc "{block1 $WORKINGDIR/block1.sdc} \
{block2 $WORKINGDIR/block2.sdc} ..."
```

or

```
propagate_constraints -block_sdc [list \
[list block1 ${WORKINGDIR}/block1.sdc] \
[list block2 ${WORKINGDIR}/block2.sdc] \] ...
```

specify_paths

```
specify_paths [-mode mode_name] [-domain clock_domain]
  [ -from {pin|port|clock|external_delay|instance}...
  | -from_rise_clock {pin|port|clock|external_delay|instance}...
  | -from_fall_clock {pin|port|clock|external_delay|instance}...
  | -from_rise_pin {pin|port|clock|external_delay|instance}...
  | -from_fall_pin {pin|port|clock|external_delay|instance}...
  [ -through {pin|port|instance}...
  | -through_rise_pin {pin|port|instance}...
  | -through_fall_pin {pin|port|instance}... ] ...
  [ -to {pin|port|clock|external_delay|instance}... | 
  | -to_rise_clock {pin|port|clock|external_delay|instance}...
  | -to_fall_clock {pin|port|clock|external_delay|instance}...
  | -to_rise_pin {pin|port|clock|external_delay|instance}...
  | -to_fall_pin {pin|port|clock|external_delay|instance}...
  [ -capture_clock_pins pin...
  | -capture_clock_pins_rise_clock pin...
  | -capture_clock_pins_fall_clock pin...
  | -capture_clock_pins_rise_pin pin...
  | -capture_clock_pins_fall_pin pin... ]
  [-lenient]
```

Creates a string that indicates the path of a particular timing exception. You must use the `specify_paths` command as an argument to the `-paths` option in any of the timing exception commands. Paths can be selected using the `-from`, `-through`, `-to`, or `-paths` options. You must provide at least one of these three options.

The `specify_paths` command gives you more detailed control when specifying timing exceptions than the `-from`, `-through`, `-to` options in these timing exceptions could provide.

Options and Arguments

`-capture_clock_pins pin`

Specifies a Tcl list of sequential clock pins that capture data. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

`-capture_clock_pins_fall_clock pin`

Specifies a Tcl list of sequential clock pins that capture data at the falling edge of the ideal clock waveform. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

Command Reference for Encounter RTL Compiler

Constraints

`-capture_clock_pins_fall_pin pin`

Specifies a Tcl list of sequential clock pins that capture data at the fall transitions of the specified sequential clock pin. This option can be useful with complex sequential cells such as RAMs that have multiple clock pins.

`-capture_clock_pins_rise_clock pin`

Specifies a Tcl list of sequential clock pins that capture data at the rising edge of the ideal clock waveform.

`-capture_clock_pins_rise_pin pin`

Specifies a Tcl list of sequential clock pins that capture data at the rise transitions of the specified sequential clock pin.

`-domain clock_domain`

Specifies a clock domain the paths should be restricted to.

`-from {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

Also, if you specify both the `-from` option on an enable pin of a latch and the `-lenient` option, the paths starting on the D pin will also be included in the timing exception.

`-from_fall_clock {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of start points for the paths. The paths are restricted to launch at the falling edge of an ideal clock waveform. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

`-from_fall_pin {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of start points for the paths. The paths are restricted to fall transitions at the start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

Command Reference for Encounter RTL Compiler

Constraints

-from_rise_clock {*pin | port | clock | external_delay | instance*}

Specifies a Tcl list of start points for the paths. The paths are restricted to launch at the rising edge of an ideal clock waveform. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

-from_rise_pin {*pin | port | clock | external_delay | instance*}

Specifies a Tcl list of start points for the paths. The paths are restricted to rise transitions at the start points. The start points can be input ports of your design, clock pins of flip-flops, clock objects, a combination of these, instances, or input ports to which specified external delay timing exceptions apply.

-lenient

Also, if you specify both the **-from** option on an enable pin of a latch and the **-lenient** option, the paths starting on the D pin will also be included in the timing exception.

If a **-from** option is provided that is not a valid start point or a **-to** option is provided that is not a valid end point, then a warning is issued but the rest of the command succeeds. These invalid points are stored in the exception, but will not affect timing analysis results. A warning is also issued when using the **report timing -lint** command in this situation.

-mode *mode_name*

Indicates the path of a particular timing exception for a specified mode.

-through {*pin | port | instance*}

Specifies a Tcl list of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the **-through** option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

Command Reference for Encounter RTL Compiler

Constraints

`-through_fall_pin {pin | port | instance}`

Specifies a Tcl list of points that a path must traverse. The path is restricted to fall transitions at the indicated points. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-through_rise_pin {pin | port | instance}`

Specifies a Tcl list of points that a path must traverse. The path is restricted to rise transitions at the indicated points. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances.

You can repeat the `-through` option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.

`-to {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of end points for the paths. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_fall_clock {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of end points for the paths. The paths are restricted to capture at the falling edge of an ideal clock waveform. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

`-to_fall_pin {pin | port | clock | external_delay | instance}`

Specifies a Tcl list of end points for the paths. The paths are restricted to fall transitions at the end points. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

Command Reference for Encounter RTL Compiler

Constraints

-to_rise_clock {pin | port | clock | external_delay | instance}

Specifies a Tcl list of end points for the paths. The paths are restricted to capture at the rising edge of an ideal clock waveform. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

-to_rise_pin {pin | port | clock | external_delay | instance}

Specifies a Tcl list of end points for the paths. The paths are restricted to rise transitions at the end points. The end points can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which specified external delay timing exceptions apply.

Examples

- The following example specifies the paths for the `path_disable` timing exception, by specifying a clock pin as start point without any other restrictions:

```
rc:/> path_disable -paths [specify_paths -from clk1]
```

- The following example creates a group containing paths that are launched by clock `clk1` and which start with a rise pin transition at their start point:

```
rc:/> path_group -paths [specify_paths -from_rise_pin clk1] -group I20
```

- The following example uses the `-to_fall_clock` option with a pin object:

```
rc:/> path_disable -paths [specify_paths -to_fall_clock ff1/D]
```

The above example specifies that the `path_disable` command should be applied to paths that end at pin `ff1/D` and are captured by a falling edge of an ideal clock waveform.

- Consider an input pin of a RAM that has setup arcs from both pin `CK1` and `CK2`. The following example applies a timing exception to paths using the setup arcs from `CK1`, but not to paths using the setup arcs from `CK2`:

```
rc:/> path_disable -paths [specify_paths -capture_clock_pins RAM/CK1]
```

Command Reference for Encounter RTL Compiler

Constraints

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*:

- [Specifying Paths for Exceptions](#)
- [Performing Multi-Mode Timing Analysis](#)

Affects these commands:	<u>multi_cycle</u> on page 304 <u>path_adjust</u> on page 308 <u>path_delay</u> on page 312 <u>path_disable</u> on page 315 <u>path_group</u> on page 318 <u>report qor</u> on page 477 <u>report timing</u> on page 498 <u>write_encounter</u> on page 243 <u>write_script</u> on page 262 <u>write_sdc</u> on page 265
Related commands:	<u>create_mode</u> on page 287 <u>define_clock</u> on page 290 <u>external_delay</u> on page 298
Affects this attribute:	<u>paths</u>

validate_constraints

```
validate_constraints [-rtl] [-netlist string]
                     [-init_sequence_file string] [-sdc string]
                     [-logfile file] [> file]
```

Validates the SDCs specified in the SDC files against the RTL or netlist. The command can be used any time after elaborating the design. If you do not specify either the *-rtl*, *-netlist*, or *-sdc* options, RTL Compiler will validate the internally generated constraints against the design at its current state.

Use this command in the False Path and Multi-cycle Path validation flows.

Options and Arguments

<i>file</i>	Specifies the name of the file to write the report.
<i>-init_sequence_file string</i>	Specifies the UNIX path to the initialization sequence file for MCP validation.
<i>-logfile string</i>	Creates a separate CCD logfile. You must specify the UNIX path to the file.
<i>-netlist string</i>	Specifies the UNIX path to the netlist. This option also indicates that the validation should happen against the netlist (as opposed to the RTL).
<i>-rtl</i>	Indicates that the validation should happen against the RTL (as opposed to the netlist).
<i>-sdc string</i>	Specifies the UNIX path to the SDC files.

Examples

- The following command validates the SDCs in the `top.sdc` file against the RTL:
`rc:/> validate_constraints -rtl -sdc /home/test/top.sdc`
- The following command validates the `generic.nl.v` netlist against the `lane.sdc`:
`rc:/> validate_constraints -netlist /home/cody/design/netlist.nl.v \
 -sdc /home/cody/lane.sdc`

Command Reference for Encounter RTL Compiler

Constraints

Related Information

[Using the Validate Flow without Dofiles](#) in *Interfacing between Encounter RTL Compiler and Encounter Conformal*

Related command: [generate_constraints](#) on page 302

Elaboration and Synthesis

- [elaborate](#) on page 332
- [get_remove_assign_options](#) on page 335
- [merge_to_multibit_cells](#) on page 336
- [remove\(assigns without optimization\)](#) on page 337
- [remove\(inserted sync enable logic\)](#) on page 340
- [retime](#) on page 341
- [set_remove_assign_options](#) on page 344
- [synthesize](#) on page 348

elaborate

```
elaborate [-parameters string] [top_module_name]...
           [-libpath path]... [-libext extension]...
```

Creates a design from a Verilog module or from a VHDL entity and architecture. Undefined modules and VHDL entities are labeled “unresolved” and treated as blackboxes.

The command returns the directory path to the top-level design(s) that it creates.

Note: Before elaborating a design, load your library using the `library` attribute and load your design using the `read_hdl` command into the rc shell.

Options and Arguments

`-libext extension` Specifies the extension of the Verilog library files.

Note: User-specified extensions overwrite the default extensions.

Default: .v and .V

`-libpath path` Specifies the search path to be used to look for unresolved Verilog module instances.

You can specify a relative path with respect to the working directory (.) or an absolute path.

Note: ~ is currently not supported.

`module` Specifies the name of the top-level module to elaborate.

If `module` is not specified, all top-level modules are elaborated.

Note: When reading in a structural file using the `read_hdl -netlist` command, use this option to specify the top module name.

`-parameters string`

Changes the values of Verilog parameters or VHDL generics that are used within the top level code to the specified values.

Specify the new values in the same sequence as the parameter definitions appear in the code to ensure proper substitution during elaboration.

RTL Compiler supports pure and sized integer specifications, and specifications of parameters for submodules and interfaces.

The following example shows a sized integer specification. In this case, 6 is the size (number of bits), d the format (decimal), and 43 the value.

```
elaborate -parameters {6'd43}
```

You can specify a parameter positionally, as an integer in the parameter list, or by name, as a two-element Tcl list. The first element is the name and the second element is the value. For example, you can do either:

```
elaborate -parameters {5 10}
```

```
elaborate -parameters {{width 5} {depth 10}}
```

The following example shows the specification of parameters for submodules or interface instances:

```
elaborate -parameters {b2.m 0 b1.w 7'd111}
```

Examples

- In the following example, the top-level code contains the following parameters in this order:

```
parameter data_width1 = 8 ;
parameter data_width2 = 12 ;
parameter averg_period = 4 ;
```

The following command changes `data_width1` to 14, `data_width2` to 12, and `averg_period` to 8 during elaboration:

```
rc:/>elaborate -parameters {14 12 8} TOP
```

- The following example reads in file `top.v` which has an instance of module `sub`, but file `top.v` does not contain a description of module `sub`.

```
set_attr hdl_search_path { ../../src ../../incl }
elaborate -libpath ../mylibs -libpath /home/verilog/libs -libext ".h"
-libext ".v" top.v
```

The latter command is equivalent to

```
elaborate -libpath { ../mylibs /home/verilog/libs } -libext { ".h" ".v" } top.v
```

First, `elaborate` looks for the `top.v` file in the directories specified through the `hdl_search_path` attribute. After `top.v` is parsed, `elaborate` looks for undefined modules (such as `sub`) in the directories specified through the `-libpath` option. First, the tool looks for a file that corresponds to the name of the module appended by the first specified file extension (`sub.h`). Next, it looks for a file that corresponds to the name of the module appended by the next specified file extension (`sub.v`), and so on.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

Related Information

Affected by this command: [read_hdl](#) on page 201

Affected by this attribute: [library](#)

get_remove_assign_options

```
get_remove_assign_options  
    [-all | -design {design|subdesign}]
```

Allows you to check which options you set for the replacement of assign statements on a design or subdesign.

If you did not set any options on a subdesign, this command will not return anything for this subdesign. However if you did set options for the design, these options will be used for all subdesigns unless you specified the `-skip hierarchical subdesigns` option.

Options and Arguments

-all Returns settings for the design and all subdesigns.

-design {*design* | *subdesign*} Returns the settings for the design or specified subdesign.

Related Information

Affected by these commands: [remove_assigns_without_optimization](#) on page 337
[set_remove_assign_options](#) on page 344

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

merge_to_multibit_cells

merge_to_multibit_cells design

Performs standalone mapping of single bit cells to multibit cells without performing any other optimizations.

Note: This command can only be run on a mapped design.

Options and Arguments

design Specifies the name of the design for which to perform multibit mapping.

Related Information

Mapping to MultiBit Cells in *Encounter RTL Compiler Synthesis Flows*

Affected by these attributes: [use_multibit_cells](#)
[use_multibit_combo_cells](#)
[use_multibit_seq_and_tristate_cells](#)

remove_assigns_without_optimization

```
remove_assigns_without_optimization
    [-buffer_or_inverter <libcell>]
    [-ignore_preserve_map_size_ok]
    [-ignore_preserve_setting]
    [-include_local_constant_assigns]
    [-respect_boundary_optimization]
    [-preserve_dangling_nets]
    [-skip_unconstrained_paths]
    [-skip_hierarchical_subdesigns]
    [-verbose]
    [-design {subdesign|design}]
```

Controls the aspects of the replacement of assign statements in the design with buffers or inverters without performing incremental optimization.

To avoid incremental optimization when removing assign statements, make sure that the `remove_assigns` attribute is set to `false`, and specify this command after synthesis.

Options and Arguments

`-buffer_or_inverter libcell`

Specifies the buffer or inverter to be used to replace the assign statements. The specified cell must be part of a library that was loaded.

If this option is omitted, RTL Compiler will determine which buffers or inverters to insert. If a particular library domain does not have any buffers or inverters, no buffers or inverters will be added in that domain. However, buffers or inverters will be added in other domains depending on availability.

`-design {design | subdesign}`

Indicates that the specified command options apply to the specified design or subdesign. If neither is specified, the options apply to the entire design.

`-ignore_preserve_map_size_ok`

Allows assign statements in the module to be removed if the `preserve` attribute on the module is set to `map_size_ok`.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

`-ignore_preserve_setting`

Allows assign statements in the module to be removed independent of the setting of the `preserve` attribute on the module.

`-include_local_constant_assigns`

Allows to remove local constant driven assigns that have multiple drivers and do not drive hierarchical ports.

`-no_buffers_use_inverters`

Specifies to search for inverters and to use them in case the library or library domain does not have any buffers. If buffers exist they will be used, irrespective of whether the library has inverters or not.

`-preserve_dangling_nets`

Specifies to preserve a net that is driven by a buffer or inverter that is inserted when assign statements are replaced and if that net does not have a fanout in the next level of hierarchy and was not driven by a constant before adding the buffer or inverter.

`-respect_boundary_optimization`

Prevents RTL Compiler from looking beyond the boundary of the subdesign, when considering subports driven by constants, if the `boundary_opto` attribute on the subdesign was set to `false`.

`-skip_hierarchical_subdesigns`

Restricts the replacement of `assign` statements to one level only. By default, the replacement of `assign` statements is done recursively.

`-skip_unconstrained_paths`

Prevents RTL Compiler from removing assigns on unconstrained paths. These paths can be paths driven by a constant, false paths with assigns, or paths without any timing constraints.

`-verbose`

Displays all messages during assign removal.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

Related Information

Affected by this attribute: [remove_assigns](#)

remove_inserted_sync_enable_logic

```
remove_inserted_sync_enable_logic [> file]
```

Removes timing critical synchronous enable logic from flops.

Tools like PowerPro™ from Calypto™ can insert synchronous enable logic for sequential clock-gating in the RTL code and write out optimized RTL. However if the synchronous enable logic is inserted on a timing critical path, RTL Compiler can remove the synchronous enable logic to improve overall timing of the design.

Use the `remove_inserted_sync_enable_logic` command after mapping and before incremental or after incremental optimization.

By disconnecting the synchronous enable, some sequential instances can become unloaded. Run incremental optimization (`synthesize -incremental`) to remove these unloaded sequential instances.

Options and Arguments

<i>file</i>	Specifies the name of the file to which RTL Compiler must write the list of flops from which the synchronous enable pins were removed.
-------------	--

Related Information

Related command: [synthesize](#) on page 348

retime

```
retime [-prepare] [-min_area] [-min_delay]
      [-effort {medium | low | high}]
      [-clock clocks]
      [design | subdesign]...
```

Improves the performance of the design by either optimizing the area or the clock period (timing) of the design. If no option is specified, the `-min_delay` option is implied.

Optimization is realized through appropriately moving registers. The area optimization will not be done at the expense of timing. That is, optimizing the area will not degrade the timing.

Options and Arguments

design | *subdesign* Specifies the name of the design or subdesign you want to retime.

`-clock clocks` Specifies to perform retiming on the registers clocked by the specified clocks

`-effort {medium | low | high}`
The effort levels are only available for the `min_delay` option.

- `high` — RTL Compiler consumes as much time as necessary to provide the optimum timing solution.
- `low` — Provides a rough retiming estimate. This effort level provides the quickest results among the three effort levels.
- `medium` — Provides results that are approximately within 1% of the optimum solution.

Default: `medium`

`-min_area`
Optimizes the design for area by minimizing the number of registers without degrading the critical path in the design.

`-min_delay`
Optimizes the design for timing. For the best results, you should first issue the `retime -prepare` command separately before issuing the `retime -min_delay` command.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

-prepare	Prepares the design for retiming and then synthesizes the design. Specifically, the <code>retime -prepare</code> command prepares the design by constraining paths according to the path delays through registers (from inputs to outputs) as opposed to register to register. There is no need to separately issue the <code>synthesize</code> command after issuing the <code>retime -prepare</code> command.
----------	---

Examples

- The following example retimes the design to optimize for timing:

```
...
rc:/> retime -prepare
rc:/> retime -min_delay
...
```

Related Information

[Retiming the Design in Using Encounter RTL Compiler](#)

Related command: [synthesize](#) on page 348

Affected by these attributes: [dont_retime](#)
[retime](#)
[retime_async_reset](#)
[retime_effort_level](#)
[retime_hard_region](#)
[retime_move_mux_loop_with_reg](#)
[retime_optimize_reset](#)
[retiming_clocks](#)

Related attributes: [retime_original_registers](#)
[retime_reg_naming_suffix](#)
[retime_verification_flow](#)
[trace_retime](#)

retime

```
retime {false | true}
```

Default: false

Read-write design attribute. Marks the specified design to be retimed during synthesis. This attribute must be set after the elaborate command but before the synthesize command.

Example

The following marks the m1 design for retiming:

```
rc:/> set_attribute retime true [find / -design m1]
Setting attribute of m1: 'retime' = true
```

Related Information

Affects this command:	synthesize
Affected by these commands:	dont_retime retime_hard_region retime_reg_naming_suffix
Related attribute:	(subdesign) retime

set_remove_assign_options

```
set_remove_assign_options [-design {design | subdesign}]
  [-buffer_or_inverter libcell]
  [-no_buffers_use_inverters]
  [-ignore_preserve_setting]
  [-ignore_preserve_map_size_ok]
  [-include_local_constant_assigns]
  [-preserve_dangling_nets]
  [-respect_boundary_optimization]
  [-skip_unconstrained_paths] [-verbose]
  [-skip_hierarchical_subdesigns]
  | -reset]
```

Controls the aspects of the replacement of assign statements in the design with buffers or inverters, which is controlled by the `remove_assigns` root attribute. The actual replacement happens during the next incremental optimization run.

The replacement of assign statements is performed

- Only on the objects (design or subdesigns) specified with the `-design` option
- Only once during each incremental optimization run operation
 - If your script contains multiple occurrences of the `set_remove_assign_options` command with different settings for the same object, the last settings before the next `synthesize` command will prevail.
- On the objects in the order of their corresponding `set_remove_assign_options` commands.

The specified options apply to all subsequent incremental optimization runs unless you reset the options with the `-reset` option.

Options and Arguments

`-buffer_or_inverter libcell`

Specifies the buffer or inverter to be used to replace the assign statements. The specified cell must be part of a library that was loaded.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

If this option is omitted, RTL Compiler will determine which buffers or inverters to insert. If a particular library domain does not have any buffers or inverters, no buffers or inverters will be added in that domain. However, buffers or inverters will be added in other domains depending on availability.

`-design {design | subdesign}`

Indicates that the specified command options apply to the specified design or subdesign. If neither is specified, the options apply to the entire design.

`-ignore_preserve_map_size_ok`

Allows `assign` statements in the module to be removed if the `preserve` attribute on the module is set to `map_size_ok`.

`-ignore_preserve_setting`

Allows `assign` statements in the module to be removed independent of the setting of the `preserve` attribute on the module.

`-include_local_constant_assigns`

Allows to remove local constant driven assigns that have multiple drivers and do not drive hierarchical ports.

`-no_buffers_use_inverters`

Specifies to search for inverters and to use them in case the library or library domain does not have any buffers. If buffers exist they will be used, irrespective of whether the library has inverters or not.

`-preserve_dangling_nets`

Specifies to preserve a net that is driven by a buffer or inverter that is inserted when `assign` statements are replaced and if that net does not have a fanout in the next level of hierarchy and was not driven by a constant before adding the buffer or inverter.

`-respect_boundary_optimization`

Prevents RTL Compiler from looking beyond the boundary of the subdesign, when considering subports driven by constants, if the `boundary_opto` attribute on the subdesign was set to `false`.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

-reset	Specifies to use the command with the default settings.
-skip_hierarchical_subdesigns	Restricts the replacement of <code>assign</code> statements to one level only. By default, the replacement of <code>assign</code> statements is done recursively.
-skip_unconstrained_paths	Prevents RTL Compiler from removing <code>assign</code> statements on unconstrained paths. These paths can be paths driven by a constant, false paths with assigns, or paths without any timing constraints.
-verbose	Displays all messages during <code>assign</code> removal.

Examples

- The following command specifies to use buffer BUFX2 to replace `assign` statements in subdesign `med`.

```
rc:/> set_remove_assign_options -buffer BUFX2 [find / -subdesign med]
```

- In the following example, the subdesigns `bottom` and `top` represent two different library domains. Two different kinds of buffers are specified for each domain.

```
rc:/> set_remove_assign_options -buffer BUFX2 [find / -subdesign bottom]
rc:/> set_remove_assign_options -buffer BUFX7 [find / -subdesign top]
```

- In the following example, several settings are specified for subdesign `sub` before the `synthesize` command.

```
set_remove_assign_options -buffer_or_inverter [ find / -libcell buf1] \
-design {find / -subdesign sub}
set_remove_assign_options -buffer_or_inverter [ find / -libcell inv] \
-design sub
synthesize -incr
```

Only the second `set_remove_assign_options` command is taken into account during the next optimization run and no `assign` removals are performed at the top level.

- In the following example, several incremental optimization runs are performed.

```
set_remove_assign_options -skip_unconstrained_paths -design top
synthesize -incr
set_remove_assign_options -reset -design top
synthesize -incr
```

During the first optimization run, unconstrained paths remain untouched in design `top`. However, before the second optimization run starts the options have been reset for design `top` and `assign` statement can now be removed from the unconstrained paths.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

Related Information

For use with the CPF flow, refer to:

[Using CPF for Multiple Supply Voltage Designs in Low Power in Encounter RTL Compiler](#)

[Using CPF for Designs Using Power Shutoff Methodology in Low Power in Encounter RTL Compiler](#)

[Using CPF for Designs Using Dynamic Voltage Frequency Scaling in Low Power in Encounter RTL Compiler](#)

Affects this command: [synthesize -incremental](#)

Affected by this attribute: [remove_assigns](#)

synthesize

```
synthesize [-effort {medium | low | high}]
           [-to_generic] [-to_mapped] [-to_placed] [-spatial]
           [[-auto_identify_shift_register]
            [-shift_register_min_length integer]
            [-shift_register_max_length integer]]
           [-incremental | -no_incremental] [design]...
```

Determines the most suitable design implementation using the given design constraints (clock cycle, input delays, output delays, technology library, and so on).

The `synthesize` command takes a list of top-level designs, synthesizes the RTL blocks, optimizes the logic, and performs technology mapping.

Options and Arguments

`-auto_identify_shift_register`

Automatically identifies functional shift register segments.

`design`

Specifies the name of the design to synthesize.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used. If multiple top-level designs exist, all are synthesized.

`-effort {medium | low | high}`

Specifies the effort to use during optimization.

Default: medium

`-incremental`

Incrementally optimizes mapped gates. Allows the mapper to preserve the current implementation of the design and perform incremental optimizations if and only if the procedure guarantees an improvement in the overall cost of the design.

Note: This option only works with an already mapped design.

`-no_incremental`

Disables incremental optimization.

`-shift_register_max_length integer`

Specifies the maximum sequential length of the shift register for auto-identification.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

Note: This option applies only when you specify
-auto_identify_shift_register.

Default: Longest detected shift register segment

`-shift_register_min_length integer`

Specifies the minimum sequential length of the shift register for auto-identification.

Note: This option applies only when you specify
-auto_identify_shift_register.

Default: 2

`-spatial`

Performs a quick placement to optimize the mapped gates.

Note: You must have access to the Encounter® place and route tool to run this command option.

`-to_generic`

Performs RTL optimization.

This is the default option if the RTL design has not been optimized yet.

`-to_mapped`

Maps the specified design(s) to the cells described in the supplied technology library and performs logic optimization. The aim of the optimization is to provide the smallest possible implementation of the synthesized design that still meets the supplied timing goal.

This is the default option when the design is in the generic or mapped state.

`-to_placed`

Performs placement and placement-based optimizations.

This option is only available with the RTL Compiler Advanced Physical Option.

Note: You must have access to the Encounter® place and route tool to run this command option.

Note: This option requires an Encounter executable of version 8.1 or later. However, it is highly recommended that you use the same versions of Encounter and RTL Compiler.

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

Examples

- The following example synthesizes and optimizes all of the top-level designs below the current position in the design hierarchy into generic logic.

```
rc:/> synthesize
```

The following table shows which actions are performed, depending on the state of the design.

Table 7-1 Actions Performed with No Option Specified

Current Design State		
RTL	Generic	Mapped
■ RTL Optimization	■ Mapping ■ Incremental Optimizations	■ Unmapping ■ Remapping ■ Incremental Optimizations
(same as -to_generic)	(same as -to_mapped)	(same as -to_mapped)

- The following example limits synthesis to a single design `main`:

```
rc:/> synthesize main
```

- The following example maps multiple designs at the same time:

```
rc:/> synthesize -to_mapped design1 design2
```

- After the following example, the design in memory will be at the Boolean (generic) level of abstraction:

```
rc:/> synthesize -to_generic
```

The following table shows the actions that are performed for the `-to_generic` option depending on the state of the design.

Table 7-2 Actions Performed with -to_generic Option Specified

Current Design State		
RTL	Generic	Mapped
■ RTL Optimization	■ nothing	■ Unmapping

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

- The following example requests mapping of the design:

```
rc:/> synthesize -to_mapped
```

The following table illustrates how the `-to_mapped` option affects the design:

Table 7-3 Actions Performed with `-to_mapped` Option Specified

Current Design State		
RTL	Generic	Mapped
■ RTL Optimization	■ Mapping	■ Unmapping
■ Mapping	■ Incremental Optimizations	■ Remap
■ Incremental Optimizations		■ Incremental Optimizations

- [Table 7-4](#) on page 351 illustrates how the `-incremental` option affects the design:

Table 7-4 Actions Performed with `-incremental` Option Specified

Options	Current Design State	
	RTL/Generic	Mapped
<code>-to_generic -incremental</code>	Disable <code>-incremental</code> and proceed	Unmapping
<code>-to_mapped -incremental</code>	Disable <code>-incremental</code> and proceed	Incremental Optimization
<code>-incremental</code>	Disable <code>-incremental</code> and proceed	Incremental Optimization

- The following command places and optimizes the design for silicon:

```
rc:/> synthesize -to_placed
```

Command Reference for Encounter RTL Compiler

Elaboration and Synthesis

The following table illustrates how the `-to_placed` option affects the design:

Table 7-5 Actions Performed with `-to_placed` Option Specified

Options	Current Design State		
	RTL	Generic	Mapped
<code>-to_placed</code>	RTL Optimization Mapping Placement Post-placement incremental optimizations	Mapping Placement Post-placement incremental optimizations	Placement Post-placement incremental optimizations
<code>-to_placed -incremental</code>	(same as <code>-to_placed</code>)	(same as <code>-to_placed</code>)	(same as <code>-to_placed</code>)

Related Information

- Affects these commands:
- [report area](#) on page 385
 - [report clock_gating](#) on page 391
 - [report datapath](#) on page 400
 - [report design_rules](#) on page 405
 - [report gates](#) on page 427
 - [report power](#) on page 462
 - [report summary](#) on page 492
 - [report timing](#) on page 498
- Related Command
- [remove_assigns_without_optimization](#) on page 337
- Affected by these attributes:
- [auto_ungroup](#)
 - [iopt_force_constant_removal](#)
 - [iopt_sequential_resynthesis](#)
 - [iopt_sequential_resynthesis_min_effort](#)

Analysis and Report

- [all_connected on page 356](#)
- [all_des on page 357](#)
- [all_des_inps on page 358](#)
- [all_des_insts on page 359](#)
- [all_des_outs on page 360](#)
- [all_des_seqs on page 361](#)
- [all_lib on page 363](#)
- [all_lib_bufs on page 364](#)
- [all_lib_ties on page 365](#)
- [analyze_library_corners on page 366](#)
- [check_design on page 368](#)
- [clock_ports on page 372](#)
- [compare_sdc on page 373](#)
- [fanin on page 374](#)
- [fanout on page 377](#)
- [report on page 380](#)
- [report_area on page 385](#)
- [report_boundary_opto on page 387](#)
- [report_cdn_loop_breaker on page 389](#)
- [report_cell_delay_calculation on page 390](#)
- [report_clock_gating on page 391](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

- [report clocks](#) on page 397
- [report congestion](#) on page 399
- [report datapath](#) on page 400
- [report design rules](#) on page 405
- [report dft_chains](#) on page 406
- [report dft_core_wrapper](#) on page 411
- [report dft_registers](#) on page 415
- [report dft_setup](#) on page 419
- [report dft_violations](#) on page 423
- [report disabled_transparent_latches](#) on page 426
- [report gates](#) on page 427
- [report hierarchy](#) on page 430
- [report instance](#) on page 432
- [report isolation](#) on page 435
- [report level_shifter](#) on page 438
- [report memory](#) on page 442
- [report memory_cells](#) on page 443
- [report messages](#) on page 444
- [report multibit_inferencing](#) on page 446
- [report net_cap_calculation](#) on page 449
- [report net_delay_calculation](#) on page 450
- [report net_res_calculation](#) on page 451
- [report nets](#) on page 452
- [report opcg_equivalents](#) on page 455
- [report operand_isolation](#) on page 456
- [report ple](#) on page 458
- [report port](#) on page 460

Command Reference for Encounter RTL Compiler

Analysis and Report

- [report power](#) on page 462
- [report power_domain](#) on page 475
- [report qor](#) on page 477
- [report scan_compressibility](#) on page 481
- [report sequential](#) on page 483
- [report slew_calculation](#) on page 486
- [report state_retention](#) on page 487
- [report summary](#) on page 492
- [report test_power](#) on page 494
- [report timing](#) on page 498
- [report units](#) on page 505
- [report utilization](#) on page 506
- [report yield](#) on page 507
- [statistics](#) on page 508
- [statistics add_metric](#) on page 509
- [statistics log](#) on page 510
- [statistics read](#) on page 512
- [statistics remove_metric](#) on page 513
- [statistics report](#) on page 514
- [statistics reset](#) on page 517
- [statistics run_stage_ids](#) on page 518
- [statistics write](#) on page 519
- [timestat](#) on page 520
- [validate_timing](#) on page 521

all_connected

```
all_connected {net [-leaf_pin] | pin | pgpin | port}...
```

If the specified object is a net, the command returns the list of all the pins connected to the net. If the object is a pin or a port, the command returns the net connected to the pin or the port.

Options and Arguments

<i>-leaf_pin</i>	Returns only the leaf cell pins connected to the specified net.
<i>net</i>	Specifies a net. The ensuing list will return a list of all the pins connected to this net.
<i>pin</i>	Specifies a pin. The ensuing list will return the net connected to the pin.
<i>pgpin</i>	Specifies a power or ground pin. The ensuing list will return the net connected to the pin.
<i>port</i>	Specifies a port. The ensuing list will return the net connected to the port.

all des

```
all des {inps | insts | outs | seqs}
```

Generates a Tcl list based on the specified object. For more information on specific `all des` commands, see [Related Information](#).

Options and Arguments

<code>inps</code>	Generates a list of all input ports of the specified clock or clock domain.
<code>insts</code>	Generates a list of all the instances in the design.
<code>outs</code>	Generates a list of all output ports of the specified clock or clock domain.
<code>seqs</code>	Generates a list of all the sequential instances in the design.

Related Information

Related commands:	all des inps on page 358
	all des insts on page 359
	all des outs on page 360
	all des seqs on page 361

all des inps

```
all des inps  
  [-clock clock...] [-clock_domains clock_domain...]  
  [design]
```

Generates a Tcl list of all input ports of the specified clock or clock domain.

Options and Arguments

<i>-clock</i> <i>clock</i>	Returns a list of input ports for the specified clock or clocks.
<i>-clock_domains</i> <i>clock_domain</i>	Returns a list of input ports for the specified clock domain or domains.
<i>design</i>	Returns a list of input ports for the specified design. If a design is not specified, the input ports for the current design will be returned.

Example

- The following example returns all the input ports for the clock named *clock1*:

```
rc:/> all des inps -clock clock1  
{/designs/PENNY/ports_in/in1[3]}{/designs/PENNY/ports_in/in1[2]}  
{/designs/PENNY/ports_in/in1[1]}{/designs/PENNY/ports_in/in1[0]}  
{/designs/PENNY/ports_in/in2[3]}{/designs/PENNY/ports_in/in2[2]}  
{/designs/PENNY/ports_in/in2[1]}{/designs/PENNY/ports_in/in2[0]}
```

all des insts

```
all des insts [-unresolved] [design]
```

Generates a Tcl list of all instances in the specified design. You can return a list of only unresolved instances by specifying the `-unresolved` option.

Options and Arguments

<i>design</i>	Returns a list of instances for the specified design. If a design is not specified, the instances for the current design will be returned.
<code>-unresolved</code>	List only unresolved instances and omit everything else.

Example

- The following example returns a list of all instances in the design named STONE:

```
rc:/> all des insts STONE
/designs/STONE/instances_hier/inst1
/designs/STONE/instances_hier/inst1/instances_comb/g1
/designs/STONE/instances_hier/inst1/instances_comb/g2
/designs/STONE/instances_hier/inst1/instances_comb/g3
/designs/STONE/instances_hier/inst1/instances_comb/g4
/designs/STONE/instances_hier/inst2
/designs/STONE/instances_hier/inst2/instances_comb/g1
/designs/STONE/instances_hier/inst2/instances_comb/g2
/designs/STONE/instances_hier/inst2/instances_comb/g3
/designs/STONE/instances_hier/inst2/instances_comb/g4
```

Command Reference for Encounter RTL Compiler

Analysis and Report

all des outs

```
all des outs
  [-clock clock...] [-clock_domains clock_domain...]
  [design]
```

Generates a Tcl list of all output ports of the specified clock or clock domain.

Options and Arguments

<i>-clock</i> <i>clock</i>	Returns a list of output ports for the specified clock or clocks. This option is to find the ports with respect to a clock waveform, not a clock input port. It is valid only after you constrain the input and output ports.
<i>-clock_domains</i> <i>clock_domain</i>	Returns a list of output ports for the specified clock domain or domains.
<i>design</i>	Returns a list of output ports for the specified design. If a design is not specified, the output ports for the current design will be returned.

Example

- The following example returns all the output ports for the clock named *clock1*:

```
rc:/> all des outs -clock clock1
{/designs/STONE/ports_out/out1[1]}{/designs/STONE/ports_out/out1[0]}
{/designs/STONE/ports_out/out2[3]}{/designs/STONE/ports_out/out2[2]}
{/designs/STONE/ports_out/out2[1]}{/designs/STONE/ports_out/out2[0]}
/designs/STONE/ports_out/out3 /designs/STONE/ports_out/out4
```

all des seqs

```
all des seqs
  [-clock clock...] [-clock_domains clock_domain...]
  [-exclude instance...]
  [-level_sensitive | -edge_triggered]
  [-no_hierarchy]
  [-data_pins] [-output_pins] [-clock_pins]
  [-master_slave] [-slave_clock_pins]
  [-inverted_output] [design...]
```

Generates a Tcl list of all the flip-flops and latches in the design. Use the `-level_sensitive` option to return only the latches in the design.

Options and Arguments

<code>-clock <i>clock</i></code>	Returns a list of sequential instances for the specified clock or clocks.
<code>-clock_domains <i>clock_domain</i></code>	Returns a list of sequential instances for the specified clock domain or domains.
<code>-clock_pins</code>	Returns the clock pins in the design.
<code>-data_pins</code>	Only returns a list of data pins.
<code><i>design</i></code>	Returns a list of sequential instances for the specified design. If a design is not specified, the sequential instances for the current design will be returned.
<code>-edge_triggered</code>	Only returns a list of flip-flops in the design.
<code>-exclude <i>instance</i></code>	Specifies a list of instances to be excluded.
<code>-inverted_output</code>	Returns sequential instances that have an inverted output (Qbar)
<code>-level_sensitive</code>	Only returns a list of latches in the design.
<code>-master_slave</code>	Returns the master slave flops.
<code>-no_hierarchy</code>	Returns sequential elements only at the top-level.
<code>-output_pins</code>	Returns the output pins in the design.
<code>-slave_clock_pins</code>	Returns the slave clock pins of master slave flops.

Examples

- The following example returns all the sequential instances for the design named REID:

```
rc:/> all des seqs REID
{/designs/REID/instances_hier/accum1/instances_seq/r_reg[1]}
{/designs/REID/instances_hier/accum1/instances_seq/r_reg[2]}
{/designs/REID/instances_hier/accum1/instances_seq/r_reg[3]}
```

- The following example returns all the data pins for the design named REID:

```
rc:/> all des seqs REID -data_pins
{/designs/cpu/instances_hier/accum1/instances_seq/r_reg[1]/pins_in/d}
{/designs/cpu/instances_hier/accum1/instances_seq/r_reg[2]/pins_in/d}
{/designs/cpu/instances_hier/accum1/instances_seq/r_reg[3]/pins_in/d}
```

- The following example returns the master slave clock with the `-master_slave` option and then the slave clock pins of that master slave clock with the `-slave_clock_pins` option. If you are using the `map_to_master_slave_lssd` attribute, you must specify it before loading any libraries.

```
rc:/> set_attribute map_to_master_slave_lssd true
rc:/> set_attribute library jess.lib
...
rc:/> all des seqs -master_slave

/designs/summers/instances_seq/flop

rc:/> all des seqs -slave_clock_pins

/designs/summers/instances_seq/flop/pins_in/t0
```

Related Information

Related attributes:

[lssd_master_clock](#)
[map_to_master_slave_lssd](#)

all lib

```
all lib {bufs | ties}
```

Generates a Tcl list of library cell objects based on the specified object. For more information on specific all lib commands, see [Related Information](#).

Options and Arguments

bufs	Generates a list of all the buffers in the loaded library.
ties	Generates a list of all the tie-cells in the loaded library.

Related Information

Related commands:	all lib bufs on page 364
	all lib ties on page 365

all lib bufs

all lib bufs

Generates a Tcl list of all the buffers in the loaded library or libraries.

Example

- The following example returns all the buffers that were defined in the loaded library:

```
rc:/> all lib bufs  
/libraries/slow/libcells/BUFX1 /libraries/slow/libcells/BUFX12  
/libraries/slow/libcells/BUFX16 /libraries/slow/libcells/BUFX2
```

all lib ties

```
all lib ties {-lo | -hi | -hilo}
```

Generates a Tcl list of all the tie-cells in the loaded library.

Options and Arguments

-hi	Returns only a list of tie-hi cells (1 value tie cells).
-hilo	Returns only a list of tie-hi-lo cells (0 and 1 value tie cells).
-lo	Returns only a list of tie-lo cells (0 value tie cells).

Example

- The following example returns a list of all tie-hi cells in the library named LUX:

```
rc:/> all lib ties -hi  
/libraries/LUX/libcells/TIEHI /libraries/LUX/libcells/ANTENNA
```

analyze_library_corners

```
analyze_library_corners
  {-libraries list | -cpf file}
  [-buffer_libcell libcell]
  [-fanout integer] [-fanin integer]
  [> file]
```

Reads in the specified multi-corner libraries and determines the slowest corner. Multi-corner libraries have the same libcells but are each characterized for a specific set of operating conditions resulting in different delay and slew values.

For each libcell the tool takes into account a given load at the input pins (specified through the number of buffers at the input) and a given load at the output pins (specified through the number of buffers at the output). Given that configuration, the tool calculates all timing arcs for the libcells for each corner and reports the average delay per corner. In addition, it reports the list of libcells whose delay exceeds the delay of the corresponding cell in the slowest library.



This command should be the only command run in the synthesis session.

Options and Arguments

<code>-buffer_libcell cell</code>	Specifies the libcell to be used as buffer.
<code>-cpf file</code>	Specifies the name of the CPF file that has the libraries for the multi corners.
<code>-fanin integer</code>	Specifies the number of buffers to consider in the fanin of the input pins of each libcell. <i>Default:</i> 2
<code>-fanout integer</code>	Specifies the number of buffers to consider in the fanout of the output pins of each libcell <i>Default:</i> 10
<code>-libraries list</code>	Specifies the list of multi-corner libraries.

Command Reference for Encounter RTL Compiler

Analysis and Report

Example

The following command reads in the libraries from the CPF file. There are 8 libraries. The report shows the average delay for each library (corner) and indicates that library XS in set4 has the largest delay.

```
analyze_library_corners -cpf test.cpf
```

Library filename	Average delay
/libraries/library_domains/set8/MF	170
/libraries/library_domains/set7/MS	176
/libraries/library_domains/set6/XF	100
/libraries/library_domains/set4/XS	352
/libraries/library_domains/set2/F	162
/libraries/library_domains/set5/S	193
/libraries/library_domains/set3/VF	115
/libraries/library_domains/set1/VS	335

The slowest library : /libraries/library_domains/set4/XS
The average delay for this lib : 352

Libcell	Library	Delay	Slowest Library	Delay
AO22XA	/lib*/library_domains/set1/VS	354	/lib*/library_domains/set4/XS	346
AND2CSXA	/lib*/library_domains/set1/VS	283	/lib*/library_domains/set4/XS	280
BUFCSXAXA	/lib*/library_domains/set1/VS	241	/lib*/library_domains/set4/XS	239
NAND3BXA	/lib*/library_domains/set1/VS	356	/lib*/library_domains/set4/XS	349
NAND3BNXA	/lib*/library_domains/set1/VS	315	/lib*/library_domains/set4/XS	311
.....				
.....				
BUFXA	/lib*/library_domains/set1/VS	241	/lib*/library_domains/set4/XS	239
AND3XA	/lib*/library_domains/set1/VS	343	/lib*/library_domains/set4/XS	338
OA22XA	/lib*/library_domains/set1/VS	348	/lib*/library_domains/set4/XS	340

Note: In the report, libraries was replaced with lib* to fit the report.

check_design

```
check_design [-lib_lef_consistency]
    [-undriven [-threshold_fanout]] [-multidriven]
    [-unloaded] [-unresolved]
    [-constant [-threshold_fanout]] [-assigns]
    [-preserved] [-report_scan_pins | -skip_scan_pins]
    [-only_user_hierarchy]
    [-vname] [-all] [design] [> file]
```

Provides information on undriven and multi-driven ports and pins, unloaded sequential elements and ports, unresolved references, constants connected ports and pins, any assign statements and preserved instances in the given design. In addition, the command can report any cells that are not in both .lib and the physical libraries (LEF files).

By default, if you do not specify an option, the `check_design` command reports a summary table with this information.

Options and Arguments

-all	Reports all the information for the design with a summary at the end.
-assigns	Reports assign statements in the design.
-constant	Reports ports and pins in the design that are connected to a constant.
<i>design</i>	Specifies the name of the design to write the report.
<i>file</i>	Specifies the name of the file to write the report.
-lib_lef_consistency	Reports the cells that are present in .lib but not in the LEF file(s) and vice versa.
-multidriven	Reports ports and pins in the design that are multi-driven.
-only_user_hierarchy	Skips the tool generated internal hierarchies (such as Mux and Adders) when performing checks and only reports checks on user-created hierarchies.
-preserved	Reports all hierarchical and leaf instances in the design for which the <code>preserve</code> attribute is set to <code>true</code> .
-report_scan_pins	Includes the scan (DFT-related) pins in the checks and reports.

Command Reference for Encounter RTL Compiler

Analysis and Report

-skip_scan_pins	Excludes the scan (DFT-related) pins from the checks and reports. Note: By default, the scan pins are included in all checks and reports.
-threshold_fanout	Filters undriven or constant pins and ports with a fanout below the specified threshold.
-undriven	Reports ports and pins in the design that are undriven.
-unloaded	Reports ports and sequential elements in the design that are unloaded.
-unresolved	Reports unresolved references and empty modules in the design.
-vname	Uses the Verilog names instead of RTL Compiler design hierarchy path names in the report.

Examples

- The following example shows a sample report given when the command is executed without any options:

```
rc:/> check_design
Checking the design.

      Check Design Report
-----
Summary
-----
      Name          Total
-----
Unresolved References           0
Empty Modules                   0
Unloaded Port(s)               0
Unloaded Sequential Pin(s)     0
Assigns                         0
Undriven Port(s)               0
Undriven Leaf Pin(s)           0
Undriven hierarchical pin(s)   1
Multidriven Port(s)            0
Multidriven Leaf Pin(s)         0
Multidriven hierarchical Pin(s) 0
Multidriven unloaded net(s)     0
Constant Port(s)                0
Constant Leaf Pin(s)            0
Constant hierarchical Pin(s)    0
Preserved leaf instance(s)      5
Preserved hierarchical instance(s) 1
Libcells with no LEF cell       601
Physical (LEF) cells with no libcell 846
```

Done Checking the design.

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports the preserved instances in your design.

```
rc:/> check_design -preserved
      Checking the design.

      Check Design Report
      ----

      Preserved instances(s)
      ----

      design 'test1' has the following preserved combinational instance(s)
      /designs/test1/instances_comb/and1
      /designs/test1/instances_comb/and2
      /designs/test1/instances_comb/and3
      /designs/test1/instances_hier/U1/instances_comb/and4
      Total number of preserved combinational instances in design 'test1' : 4
      design 'test1' has the following preserved sequential instance(s)
      /designs/test1/instances_seq/dff1
      Total number of preserved sequential instances in design 'test1' : 1
      design 'test1' has the following preserved hierarchical instance(s)
      /designs/test1/instances_hier/U1
      Total number of preserved hierarchical instances in design 'test1' : 1

      Done Checking the design.
```

- The following command reports the cells which are only in .lib or in the physical library (LEF files).

```
rc:/> check_design -lib_lef_consistency
      Checking the design.

      Check Design Report
      ----

      Libcells with no corresponding LEF
      ----
      /libraries/mylib.slow/libcells/ACCSHCINX2
      ...
      /libraries/mylib.slow/libcells/p_SMDFFHQX8
      Total number of cell(s) with only library (.lib) info : 601

      LEF cells with no corresponding libcell
      ----
      /libraries/physical_cells/libcells/AN2D0
      ...
      /libraries/physical_cells/libcells/XOR4D4
      Total number cell(s) with only physical (LEF) Info : 846

      Done Checking the design.
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports all undriven pins including the DFT-related pins (highlighted in the report below).

```
rc:/> check_design -undriven
      Checking the design.

      Check Design Report
      -----
      Undriven Port(s)/Pin(s)
      -----
      No undriven combinational pin in design 'test1'

      The following sequential pin(s) in design 'test1' are undriven
      /designs/test1/instances_seq/dff1/pins_in/SE
      /designs/test1/instances_seq/dff1/pins_in/SI
      Total number of sequential undriven pins in design 'test1' : 2

      The following hierarchical pin(s) in design 'test1' are undriven
      /designs/test1/instances_hier/U1/pins_in/A      (fanout : 0)
      Total number of hierarchical undriven pins in design 'test1' : 1

      No undriven port in 'test1'

      Done Checking the design.
```

- The following command reports all undriven pins *excluding* the DFT-related pins.

```
rc:/> check_design -undriven -skip_scan_pins
      Checking the design.

      Check Design Report
      -----
      Undriven Port(s)/Pin(s)
      -----
      No undriven combinational pin in 'test1'

      No undriven sequential pin in 'test1'

      The following hierarchical pin(s) in design 'test1' are undriven
      /designs/test1/instances_hier/U1/pins_in/A      (fanout : 0)
      Total number of hierarchical undriven pins in design 'test1' : 1

      No undriven port in 'test1'

      Done Checking the design.
```

Related Information

[Run DFT Rule Checker and Floating Nets and X-Source Checks in Design for Test in Encounter RTL Compiler](#)

clock_ports

`clock_ports [design]`

Returns input ports of your design that are clock inputs.

Note: Only input ports at the top level are listed. Gated clocks and clock pins that are present in the hierarchical design internally (typical PLL outputs) will not be identified.

Note: This command is useful when you are working with an unfamiliar design.

Options and Arguments

design Specifies the design for which you want to list the clock input ports.

Examples

- The following example finds all of the clock ports of a design:

```
rc:/> clock_ports  
/designs/alu/ports in/clock
```

- In the following example, the `clock_ports` command is embedded within a `define_clock` command to apply a clock waveform to all clock input ports of the design:

```
rc:/> define_clock -period 3000 -name clock1 [clock_ports]
```

Related Information

Affected by this command: [define_clock](#) on page 290

compare_sdc

```
compare_sdc [-design string] [-rtl | -netlist file]
             -golden_sdc files [-revised_sdc files]
             [-logfile file] [-detail] [> file]
```

Compares two (or two sets of) SDC files for a design and generates a report containing differences.

To run this command you need to have access to the Encounter® Conformal® Constraint Designer (CCD) software.

Options and Arguments

<code>-design <i>string</i></code>	Specifies the top-level design in RTL Compiler.
<code>-detail</code>	Requests a detailed comparison report.
<code><i>file</i></code>	Specifies the file to which the report must be written.
<code>-golden_sdc <i>files</i></code>	Specifies the UNIX path to the golden (original) SDC files.
<code>-logfile <i>file</i></code>	Specifies the name of the CCD logfile. You must specify the UNIX path to the file.
<code>-netlist <i>file</i></code>	Specifies the UNIX path to the netlist. By default, the tool uses the RTL.
<code>-revised_sdc <i>files</i></code>	Specifies the UNIX path to the revised SDC files. If this option is not specified, the tool internally generates an SDC file for the current state of the design and uses this file for the comparison.
<code>-rtl</code>	Specifies to use the RTL as input.

Related Information

[Comparing SDC Constraint Files in Interfacing between Encounter RTL Compiler and Encounter Conformal](#)

Related command: [write_do_ccd compare_sdc](#) on page 228

fanin

```
fanin {pin | pgpin | port | subport}...
  [-min_logic_depth integer]
  [-max_logic_depth integer]
  [-min_pin_depth integer] [-max_pin_depth integer]
  [-startpoints] [-structural [-timing_model_comb]]
  [-gui] [-vname]
```

Returns pins and ports in the fanin cone for the specified objects (pins or ports) starting from the nearest timing startpoints based on the timing arcs of the libcells of the instances. Use the -structural option to do a connectivity trace (ignoring timing arcs and functions) with every output of an instance having a “structural connection” to every input and vice-versa.

Options and Arguments

`-gui` Highlights the instances along the fanin cone in the Physical Viewer. The instance at the start of the fanin cone is marked in yellow, the other instances are marked in red.

Note: The GUI should be enabled to have any effect.

`-max_logic_depth integer` Specifies the maximum number of logic levels to go back to report the fanin cone information.

Default: Infinity

`-max_pin_depth integer` Specifies the maximum number of pin levels to go back to report the fanin cone information.

Default: Infinity

`-min_logic_depth integer` Specifies the minimum number of logic levels to go back to report the fanin cone information.

Default: 0

`-min_pin_depth integer` Specifies the minimum number of pin levels to go back to report the fanin cone information.

Default: 0

Command Reference for Encounter RTL Compiler

Analysis and Report

{pin | pgpin | port | subport}

Specifies the name of a pin, pgpin, port, or subport for which you want the fanin cone information.

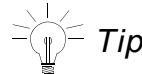
-startpoints

Returns only timing start points in the fanin cone.

-structural

Specifies to perform a connectivity-based structural trace instead of the default timing trace based on timing arcs.

Note: If there are missing timing arcs, for example, when using the SDC set_case_analysis command, the traces may report different results.



Tip
Use this option with care as it can increase runtime considerably.

-timing_model_comb

Specifies to trace through combinational timing model libcells. You can only specify this option with the -structural option.

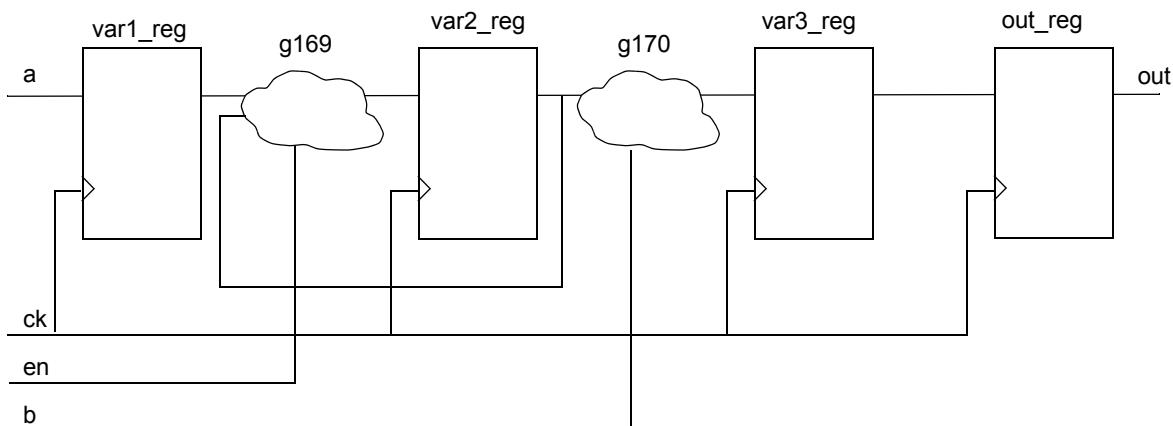
-vname

Specifies to return the path to the pins and ports in Verilog style.

Note: This option does not apply to subports and constants.

Examples

- Consider the design below.



The following example returns all the pins in the fanin cone for port `out`:

```
rc:/> fanin out  
/designs/test/instances_seq/out_reg/pins_out/Q /designs/test/instances_seq/  
out_reg/pins_in/CK
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example specifies to only return the start point for port `out` shown in the design above:

```
rc:/> fanin -startpoints out  
/designs/test/instances_seq/out_reg/pins_in/CK
```

- The following example executes a path disable from all the start points that fan out to `reg1/D`:

```
rc:/> path_disable -from [fanin -startpoint reg1/D]
```

- The following example queries the fanin of the output pin `S` of the combinational instance `adder` shown in Figure .

```
rc:/> fanin -startpoints S
```

In this case, the command returns input port `IN` and clock pin `CK`

- Use the `ls -dir` command to format the output:

```
rc:/designs/malexander/ports_in> ls -dir [fanin in1[0]]  
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_in/A  
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_out/Y  
/designs/malexander/ports_out/out1[1]  
/designs/malexander/ports_out/out3
```

- Use the `ls -dir` command with the redirect arrow to redirect the output to the specified file:

```
rc:/designs/malexander/ports_in> ls -dir [fanin in1[0]] > project.txt
```

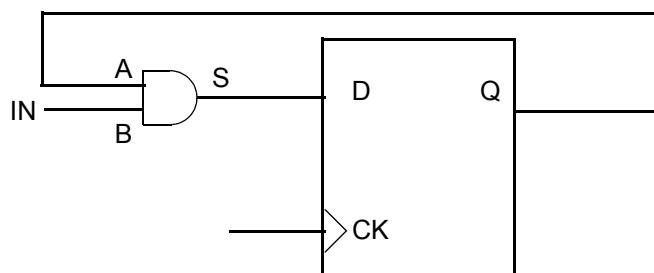
You can also use the append arrows ("`>>`").

- The following example queries the fanin of the output pin `S` of the combinational instance shown in Figure 8-1, but without using the `-startpoint` option.

```
rc:/> fanin S
```

In this case, the command returns in addition to the input port `IN` and clock pin `CK`, pins `A` and `Q`.

Figure 8-1 Example Design for fanin



fanout

```
fanout {pin | pgpin | port | subport}...
    [-min_logic_depth integer]
    [-max_logic_depth integer]
    [-min_pin_depth integer] [-max_pin_depth integer]
    [-endpoints] [-structural [-timing_model_comb]]
    [-gui] [-vname]
```

Returns pins and ports in the fanout cone for the specified objects (pins or ports) stopping at the nearest timing endpoints based on the timing arcs of the libcells of the instances. Use the -structural option to do a connectivity trace (ignoring timing arcs and functions) with every output of an instance having a “structural connection” to every input and vice-versa.

Options and Arguments

-endpoints	Returns only timing end points in the fanout cone.
-gui	Highlights the instances along the fanout cone in the Physical Viewer. The instance at the start of the fanout cone is marked in yellow, the other instances are marked in red. Note: The GUI should be enabled to have any effect.
-max_logic_depth <i>integer</i>	Specifies the maximum number of logic levels to go back to report the fanout cone information. <i>Default:</i> Infinity
-max_pin_depth <i>integer</i>	Specifies the maximum number of pin levels to go back to report the fanout cone information. <i>Default:</i> Infinity
-min_logic_depth <i>integer</i>	Specifies the minimum number of logic levels to go back to report the fanout cone information. <i>Default:</i> 0
-min_pin_depth <i>integer</i>	Specifies the minimum number of pin levels to go back to report the fanout cone information. <i>Default:</i> 0

Command Reference for Encounter RTL Compiler

Analysis and Report

{pin | pgpin | port | subport}

Specifies the name of a pin, pgpin, port, or subport for which you want the fanout cone information.

-structural

Specifies to perform a connectivity-based structural trace instead of the default timing trace based on timing arcs.

Note: If there are missing timing arcs, for example, when using the SDC set_case_analysis command, the traces may report different results.



Tip
Use this option with care as it can increase runtime considerably.

-timing_model_comb

Specifies to trace through combinational timing model libcells. You can only specify this option with the -structural option.

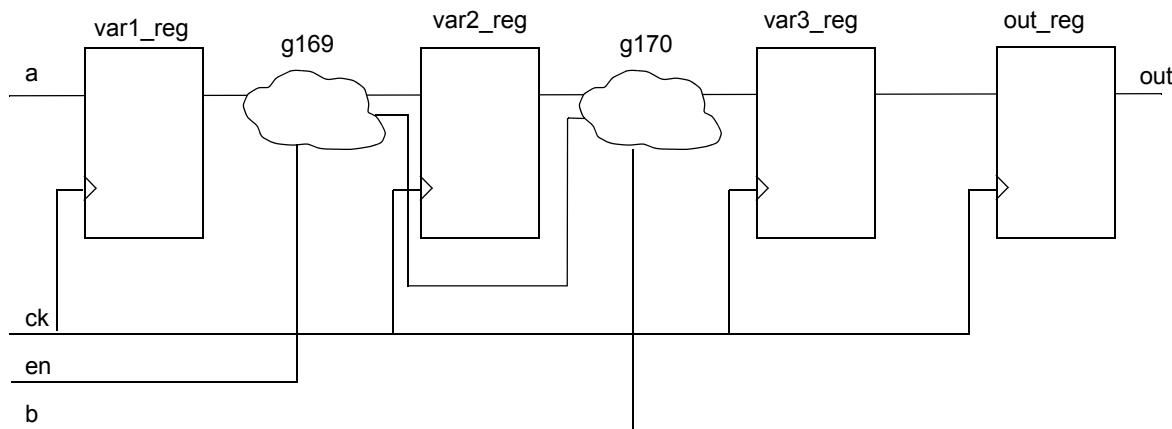
-vname

Specifies to return the path to the pins and ports in Verilog style.

Note: This option does not apply to supports and constants.

Examples

- The following example returns the pins in the fanout cone of port en in the design below:



```
rc:/> fanout en
/designs/test/instances_comb/g169/pins_out/Z /designs/test/instances_seq/
var2_reg/pins_in/D /designs/test/instances_comb/g170/pins_in/SL /designs/
test/instances_comb/g170/pins_out/Z /designs/test/instances_seq/var3_reg/
pins_in/D
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example executes a path disable on all the endpoints to which reg1/CK fans out:

```
rc:/> path_disable -to [fanout -endpoints reg1/CK]
```

- The following example returns all pins in the fanout cone up to two logic levels forward from the specified pin:

```
rc:/> fanout -max_logic_depth 2 B  
/designs/top/instances_hier/m1/instances_comb/g2/pins_in/in_0
```

- Use the `ls -dir` command to format the output:

```
rc:/designs/malexander/ports_in> ls -dir [fanout in1[0]]  
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_in/A  
/designs/malexander/instances_hier/inst1/instances_comb/g43/pins_out/Y  
/designs/malexander/ports_out/out1[1]  
/designs/malexander/ports_out/out3
```

- Use `ls -dir` with the redirect arrow to redirect the output to the specified file:

```
rc:/designs/malexander/ports_in> ls -dir [fanout in1[0]] > malexander.txt
```

You can also append arrows (">>").

report

```
report {area | boundary_opto | cdn_loop_breaker |
| cell_delay_calculation | clock_gating | clocks
| congestion | cwd | datapath | design_rules
| dft_chains | dft_registers | dft_setup
| dft_violations | disabled_transparent_latches
| gates | hierarchy | instance | isolation
| level_shifter | memory | memory_cells | messages
| multibit_inferencing
| net_cap_calculation | net_delay_calculation
| net_res_calculation | nets | operand_isolation
| ple | port | power | power_domain | qor
| scan_compressibility | sequential
| slew_calculation | state_retention | summary
| test_power | timing | units | utilization | yield}
```

Generates the specified report on synthesis results. For more information, see the [Related Information](#).

All reports have a header which contains information about the version of the tool used to generate the report, the time that the report was generated, the module for which the report is generated, the technology libraries used for synthesis, the operating conditions. The next two lines in the header depend on the setting of some attributes.

In addition, if you are running synthesis using wireload models (`interconnect_mode` attribute set to `wireload`), the header has a `Wireload mode` entry that corresponds to the value of the `wireload_mode root` attribute. In this case, RTL Compiler uses the cell area specified in the timing libraries. Consequently, the `Area mode` entry is set to `timing library`.

If you are running any of the physical synthesis flows (`interconnect_mode` attribute set to `ple`), the header has an `Interconnect mode` entry which can have the following values:

- `global`—report generated before synthesis is run (all physical flows) or generated after the design is synthesized without placement information (simple PLE flow)
- `spatial`—report generated after the design is synthesized using a fast placement (spatial flow).
- `placement`—report generated after the design is synthesized using detailed placement information (RC-P flow)

When running a physical flow, RTL Compiler uses the cell area specified in the LEF libraries and therefore `Area mode` is set to `physical library`.

You can automatically write out a gzip compressed report file. For example:

```
report port sample.gz
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Note: The report memory command does not support the gzip compressed output.

Options and Arguments

area	Reports the area of the synthesized and mapped design.
boundary_opto	Reports a summary of the boundary optimization.
cdn_loop_breaker	Reports the loop breaker buffers added by the tool and breaks the combinational loops for timing analysis.
cell_delay_calculation	Reports how the cell delay of a libcell instance is calculated.
clock_gating	Reports clock-gating information for the design.
clocks	Generates a report on the clocks of the current design.
congestion	Reports the congestion summary.
cwd	Generates a ChipWare Developer report.
datapath	Reports datapath operators that were inferred from the design.
design_rules	Reports the design rule violations.
dft_chains	Reports the scan chain data for the design.
dft_registers	Reports the scan status of all flip-flops in the design.
dft_setup	Reports the DFT setup information.
dft_violations	Reports the DFT violations.
disabled_transparent_latches	Reports disabled transparent latches.
gates	Generates a gates report.
hierarchy	Reports the design hierarchy information.
instance	Generates a report on the instances of the current design.
isolation	Reports isolation cell information for the design.
level_shifter	Reports level-shifter information for the design.
memory	Reports the memory usage in the compilation environment.
memory_cells	Reports the memory cells in the library.

Command Reference for Encounter RTL Compiler

Analysis and Report

messages	Generates a summary of error messages that have been issued.
multibit_inferencing	Reports on multibit cells in the design or library.
net_cap_calculation	Reports how the capacitance of the net is calculated.
net_delay_calculation	Reports how the net delay is calculated.
net_res_calculation	Reports how the resistance of the net is calculated.
nets	Generates a report on the nets of the current design.
opcg_equivalents	Reports the OPCG-equivalency mappings.
operand_isolation	Reports operand-isolation information for the design.
ple	Reports physical layout estimation data
port	Generates reports on the ports of the current design.
power	Generates a power leakage report.
power_domain	Generates a report with power domain information.
qor	Generates a QoR report.
scan_compressibility	Reports the scan compressibility of a design.
sequential	Generates a report on the sequential elements of the design.
slew_calculation	Reports how the slew on the driver pin of a libcell instance is calculated.
state_retention	Reports state retention information for the design.
summary	Generates an area, timing, and design rule violations report.
test_power	Reports estimated power of design during scan test
timing	Generates a timing report.
units	Reports the units used in the libraries.
utilization	Reports the floorplan utilization for the design.
yield	Generates a yield report.

Related Information

Related commands:

[report area](#) on page 385
[report boundary_opto](#) on page 387
[report cdn_loop_breaker](#) on page 389
[report cell_delay_calculation](#) on page 390
[report clock_gating](#) on page 391
[report clocks](#) on page 397
[report congestion](#) on page 399
[report datapath](#) on page 400
[report design_rules](#) on page 405
[report dft_chains](#) on page 406
[report dft_registers](#) on page 415
[report dft_setup](#) on page 419
[report dft_violations](#) on page 423
[report disabled_transparent_latches](#) on page 426
[report gates](#) on page 427
[report hierarchy](#) on page 430
[report instance](#) on page 432
[report isolation](#) on page 435
[report level_shifter](#) on page 438
[report memory](#) on page 442
[report memory_cells](#) on page 443
[report messages](#) on page 444
[report multibit_inferencing](#) on page 446
[report net_cap_calculation](#) on page 449
[report net_delay_calculation](#) on page 450
[report net_res_calculation](#) on page 451
[report nets](#) on page 452

Command Reference for Encounter RTL Compiler

Analysis and Report

[report opcg equivalents](#) on page 455
[report operand isolation](#) on page 456
[report ple](#) on page 458
[report port](#) on page 460
[report power](#) on page 462
[report power domain](#) on page 475
[report qor](#) on page 477
[report scan compressibility](#) on page 481
[report sequential](#) on page 483
[report slew calculation](#) on page 486
[report state retention](#) on page 487
[report summary](#) on page 492
[report test power](#) on page 494
[report timing](#) on page 498
[report units](#) on page 505
[report utilization](#) on page 506
[report yield](#) on page 507

Command Reference for Encounter RTL Compiler

Analysis and Report

report area

```
report area
  [-depth integer [-instance_hierarchy instance]]
  [-min_count integer] [-physical]
  [-summary] [design]... [> file]
```

Reports the following information:

- The total count of cells mapped against the hierarchical blocks in the current design
- The combined cell area in each of the blocks and the top level design (hierarchical breakup)

The Cell Area numbers are based on the cell implementations taken from the technology library after mapping. However, in PLE mode, the numbers are based on the information in the LEF libraries. It might be 0 if the information is missing in the LEF libraries.

- The Net Area refers to the estimated post-route net area and is only meaningful if you read in the LEF libraries. Net area is based on the minimum wire widths defined in the LEF and capacitance table files and the area of the design blocks.
- The wireload model adopted for each of the blocks

Note: The units used in the report depend on the units used in the library.

Options and Arguments

-depth <i>integer</i>	Specifies the number of levels of recursion.
<i>design</i>	Specifies the design for which you want to generate a report. You can also cd into the particular design directory and generate the report.
<i>file</i>	Specifies the name of the file to which to write the report.
-instance_hierarchy <i>instance</i>	Reports the area of the leaf instances in the specified hierarchical instance. This option must be specified together with the -depth option.
-min_count <i>integer</i>	Specifies the minimum instance count per block.
-physical	Specifies to use the LEF cell width and height values to calculate the area.
-summary	Prints the area summary for the design.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example generates the area report for the current design:

```
rc:/> report area
=====
Generated by:          RTL Compiler (RC) version
Generated on:          Current date Current time
Module:                mult_bit_muxed_add
Technology library:   tutorial_
Operating conditions: nominal_ (balanced_tree)
Wireload mode:         enclosed_
Area mode:             timing library
=====

***** Area *****

Instance      Cells  Cell Area  Net Area      Wireload
-----
mult_bit_muxed_add    6       69        0 suggested_10K (S)
ma1                  3       35        0 suggested_10K (S)
ma0                  3       35        0 suggested_10K (S)

(S) = wireload was automatically selected
```

- The following command prints the area summary.

```
rc:/> report area -summary
=====
...
=====

Instance      Cells  Cell Area  Net Area      Wireload
-----
mult_bit_muxed_add    6       69        0 suggested_10K (S)
```

- The following command reports the area for the leaf instances of instance ma1.

```
rc:/> report area --inst_hier ma1 -depth 2
=====
...
=====

Instance      Cells  Cell Area  Net Area      Wireload
-----
ma1          3       35        0 suggested_10K (S)
g53          0       12        0 suggested_10K (S)
g52          0       12        0 suggested_10K (S)
g51          0       10        0 suggested_10K (S)

(S) = wireload was automatically selected
```

Related Information

Analyzing the Results in [Simple PLE Flow](#), [Spatial Flow](#), and [RC-P Flow](#) in *Design with RTL Compiler Physical*

Affected by this command: [synthesize](#) on page 348

Affected by this attribute [shrink_factor](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report boundary_opto

```
report boundary_opto [instance]... [-hierarchy instance]
```

Reports a summary of the boundary optimization done on the design.

This is a summary of boundary changes on the hierarchical pins.

Options and Arguments

-hierarchy *instance*

Reports the boundary optimization done on all instances below the specified instance.

instance

Reports the boundary optimization done on the specified instance.

Example

Consider the following input RTL:

```
module top(in1,in2,out,out1,out2);
    input in1,in2;
    output out,out1,out2;
    child inst(in1,in2,out,out1,out2);
endmodule

module child(a,b,out,out1,out2);
    input a,b;
    output out,out1,out2;
    and u1(n_1,b,a);
    assign out = n_1;
    assign out1 = ~a;
    assign out2 = ~n_1;
endmodule
```

After the `synthesize -to_map` command, report the boundary optimization.

```
rc:/> report boundary_opto
=====
...
=====
Instance   Pin           Boundary Change
-----
inst      out1  routed opposite signal through 'inst/a' around 'inst'
          out2  pushed opposite signal through 'inst/out'
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

Related attribute: [boundary_change](#)

report cdn_loop_breaker

```
report cdn_loop_breaker [-sdcfile string]
                         [-version string] [design] [> file]
```

Reports the loop breaker buffers that were added by the timing engine to break the combinational loops during timing analysis.

The report lists for every loop breaker the instance name, and the driver and the load of the loop breaker.

Options and Arguments

<i>design</i>	Specifies the design for which you want the report.
<i>file</i>	Redirects the report to the specified file.
<i>-sdcfile string</i>	Creates an SDC file with the appropriate <code>set_disable_timing</code> and <code>set_false_path</code> settings.
<i>-version string</i>	Specifies a particular SDC version for the SDC file created with the <i>-sdcfile</i> option. The available versions are: 1.1, 1.3, 1.4, 1.5 or 1.7. <i>Default:</i> 1.7.

Example

The following report shows the loop breakers inserted in design *loop*.

```
rc:/> report cdn_loop_breaker
=====
Generated by:           version
Generated on:          date
Module:                loop
Technology library:    tutorial 1.1
Operating conditions: typical_case (balanced_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====
CDN Loop breaker      Driver   Load
-----
inst1/cdn_loop_breaker inst1/i1/Y i0/B
```

Related Information

Related command: [remove_cdn_loop_breaker](#) on page 974

report cell_delay_calculation

```
report cell_delay_calculation
  -from_pin pin  -to_pin pin
  [-from_rise] [-from_fall]
  [-to_rise] [-to_fall] [> file]
```

Reports how the cell delay is calculated from the look up table in the loaded technology library. Specify the cell by choosing its the driving and loading pins. The formula for calculating the delay is provided at the bottom of the report.

Options and Arguments

<i>file</i>	Redirects the report to the specified file.
<i>-from_pin pin</i>	Specifies the driving pin.
<i>-from_fall</i>	Uses the fall delay calculation from the driving pin.
<i>-from_rise</i>	Uses the rise delay calculation from the driving pin.
<i>-to_fall</i>	Uses the fall delay calculation of the loading pin.
<i>-to_pin pin</i>	Specifies the loading pin.
<i>-to_rise</i>	Uses the rise delay calculation of the loading pin.

report clock_gating

```
report clock_gating [-instance hier_instance]
  [ -preview [-gated_ff]
    [-clock clock_list | -clock_pin {pin|port|subport}...]
  | [-gated_ff] [-ungated_ff] [-no_hierarchical]
  | [-summary] [-detail]
  | [-fanout_histogram [-step {{start stop}...}]}
  | {-clock clock_list |-clock_pin {pin|port|subport}...}
    [-detail]
  | -cg_instance cg_instance...
  | -multi_stage ] [design] [> file]
```

Reports clock-gating information for the design.

Note: After importing third-party clock-gating logic, this clock-gating logic will be reported as “RC Clock Gating Instances.”

Options and Arguments

-cg_instance *cg_instance*

Reports detailed clock-gating information for the specified clock-gating instances. Information includes the library cell used for the clock-gating cell, the clock-gating style, the signals connected to the inputs and outputs of the gating logic, and the flip-flops gated by this gating cell.

A clock-gating instance is the hierarchical instance with the clock-gating logic inside.

-clock *clock_list* Limits the report to the specified clocks. The specified clocks must have been defined through either the `define_clock` command or through the SDC constraints.

-clock_pin {*pin* | *port* | *subport*}

Limits the report to the specified clock pins. Use this option if you did not define the clocks. You can specify clock pins, clock ports or clock subports.

Note: If both `-clock` and `-clock_pin` options are specified, the `-clock` option takes precedence.

design Specifies the design for which you want to generate the report.

Command Reference for Encounter RTL Compiler

Analysis and Report

-detail	Reports detailed clock-gating information. Lists all the clock-gating instances inserted, including the library cell used for the clock-gating cell, the clock-gating style, the signals connected to the inputs and outputs of the gating logic, and the toggle reduction achieved. In addition, it lists for each clock-gating instance the names of the gated flip-flops with the register efficiency. If you specify only this option, the return value of this command is the total number of clock-gating logic inserted in the design.
-fanout_histogram	Prints for a set of fanout ranges, the number of clock-gating instances with a fanout in that range, and the total number of flip-flops that these clock-gating instances gate.
<i>file</i>	Specifies the name of the file to which to write the report.
-gated_ff	Reports all the flip-flops that are clock gated and the clock-gating instances that gate the flip-flops. In addition, it lists for all gated flip-flops the register efficiency. If you specify only this option, the return value of this command is the total number of flip-flops that are gated in the design. If you specify this option with the -preview option, the report adds for each combination of clock and enable inputs (listed in the report) the names of the flip-flops that can potentially be gated.
-instance <i>hier_instance</i>	Prints the clock-gating report for the specified hierarchical instance.
-multi_stage	Shows the clock-gating instance hierarchy. This option cannot be combined with any other options.
-no_hierarchical	Limits the clock-gating information to the current module. By default, the report command traverses the hierarchy starting from the current module and reports all the clock gating found in the current module and its children modules.
-preview	Shows the potential reduction in the clock toggling when clock-gating logic would be inserted—without making any modifications to the netlist. Use this option on an unmapped or partially mapped design.

Command Reference for Encounter RTL Compiler

Analysis and Report

If you did not define the clocks, you must specify the `-clock_pin` option to identify a clock.

Note: If both `-clock` and `-clock_pin` options are specified, the `-clock` option takes precedence.

`-step {{start stop}...}`

Specifies user-defined ranges for the fanout histogram. Specify a list of lists. Each list defines a range. The stop point of the range cannot be smaller than the start point. In addition, the start point of each range must be larger than the stop point of the previous range.

The tool will report on the defined ranges and the uncovered ranges.

Note: This option can only be specified with the `-fanout_histogram` option.

`-summary`

Prints a summary report of the clock gating inserted in the design that includes the number of clock-gating instances, the number of gated flip-flops and ungated flip-flops, and for the gated flip-flops, the average toggle saving (in percent).

Note: The first two lines refer to the *leaf* clock-gating instances (RC and non-RC) that were added. If multi-stage clock gating is present, two more lines are added to the top of the summary reporting the multi-stage clock gating instances (RC and non-RC).

If used with other options, a summary report is printed at the end.

If you specify only this option, the return value of this command is the total number of clock-gating logic inserted in the design.

If no other options specified, you will get a summary report by default.

`-ungated_ff`

Reports all the flip-flops that are not clock gated, and lists whether the flop was excluded for clock-gating or not.

If you specify only this option, the return value of this command is the total number of flip-flops that are not gated in the design.

When you specify this option with the `-detail` option, the report lists the specific reason why the flip-flops were not gated.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

The following reports show output generated after clock gating has been inserted.

- The following command reports all the flip-flops that are clock gated. In this case, the return value of the command is 8.

```
rc:/> report clock_gating -gated_ff
=====
...
=====

Gated Flip-flops
-----
Module  Clock Gating Instance  Origin  Fanout  Gated Flip-flops  Register Efficiency
-----
alu      RC_CG_HIER_INST0      RC       8        aluout_reg[0]    89.000
                                                 aluout_reg[1]    89.000
...
                                                 aluout_reg[6]    89.000
                                                 aluout_reg[7]    87.000
-----
Total    1                      8
-----
```

8

- The following command gives the reason why the flip-flops remain ungated. In this case, there is no return value.

```
rc:/> report clock_gating -ungated -detail
=====
...
=====

Ungated Flip-flops With Detail
-----
Flops offering no power saving : 4
/designs/test/instances_hier/my_ff/instances_seq/q_reg[0]
/designs/test/instances_hier/my_ff/instances_seq/q_reg[2]
/designs/test/instances_hier/my_ff/instances_seq/q_reg[3]
/designs/test/instances_hier/my_ff/instances_seq/q_reg[1]
```

- The following command shows the fanout histogram:

```
rc:/> report clock_gating -fanout_histogram
=====
...
=====

CG Fanout Histogram
-----
Fanout-Size          Num-CGs   Total-FFs
-----
1 to 3               0         0
4 to 15              2         16
16 to 63              0         0
64 to 255             0         0
256 and higher        0         0
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the fanout histogram with user-defined ranges:

```
rc:/> report clock_gating -fanout_histogram -step {{1 6} {7 15}}
=====
...
=====

CG Fanout Histogram
-----
  Fanout-Size      Num-CGs      Total-FFs
-----
  1 to 6          0            0
  7 to 15         2            16
  16 and higher   0            0
```

- The following command generates detailed clock-gating information for the specified clock-gating instance:

```
rc:/> report clock_gating -cg_instance [find / -inst RC_CG_HIER_INST0]
=====
...
=====

Detail
-----
Clock Gating Instance : RC_CG_HIER_INST0
-----
  Libcell:           TLATNTSCAX2 (slow)
  Style:             latch_posedge_precontrol
  Module:            alu (alu)
  Type:              Leaf level CG Instance
  Inputs:
    ck_in       =   clock (/designs/alu/ports_in/clock)
                    TCF = (0.75000, 0.571429/ns)
    enable      =   ena (/designs/alu/ports_in/ena)
                    TCF = (0.49870, 0.020833/ns)
    test        =   LOGIC0
  Outputs:
    ck_out      =   rc_gclk
                    TCF = (0.34510, 0.333333/ns)
  Toggle Reduction = 42.00
  Gated FFs:
    Module  Clock Gating Instance  Origin  Fanout  Gated Flip-flops  Register Efficiency
    -----
    alu      RC_CG_HIER_INST0     RC      8       aluout_reg[0]    89.000
                                         aluout_reg[1]    89.000
    ...
                                         aluout_reg[6]    89.000
                                         aluout_reg[7]    87.000
    -----
    Total    1                      8
```

1

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports the number of flip-flops that are clock gated by the specified clock.

```
rc:/> report clock_gating -clock [find / -clock clock]
Info: Since -clock option is specified, options other than -detail are ignored.
Multi-stage clock gating for '/designs/alu/ports_in/clock'
=====
Max stage:    1
Total FFs with 0 stage of CG:   1
Total FFs with 1 stage of CG:   8
=====
Total FF:    9
```

Related Information

[Previewing the Effect of Clock Gating in Low Power in Encounter RTL Compiler](#)

[Reporting Clock-Gating Information in Low Power in Encounter RTL Compiler](#)

[Clock Gating Cell Specification in the Library Guide for Encounter RTL Compiler.](#)

Affected by this command: [synthesize](#) on page 348

Affected by these attributes: [lp_clock_gating_add_obs_port](#)

[lp_clock_gating_add_reset](#)

[lp_clock_gating_cell](#)

[lp_clock_gating_control_point](#)

[lp_clock_gating_exclude](#)

[lp_clock_gating_style](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report clocks

```
report clocks
  [-ideal] [-generated]
  [clock]... [-mode mode_name] [> file]
```

Generates a report on the clocks of the current design. Reports the clock period, rise, fall, domain, setup uncertainty, latency, clock ports or sources in the current design.

Use the `-generated` option to report generated clock information, and use the `-ideal` option to report an ideal clock - clock relationship.

Options and Arguments

<code>clock</code>	Specifies the name of the clock for which you want to generate the report.
<code>file</code>	Specifies the name of the file to which to write the report.
<code>-generated</code>	Adds generated clock information to the description, uncertainty, and the relationship table.
<code>-ideal</code>	Reports a clock description with the ideal clock - clock relationship.
<code>-mode mode_name</code>	Generates a report by mode on the clocks of the current design.

Example

The following example generates a clock report with generated clock information added to the table:

```
rc:/> report clocks -generated
=====
Generated by:          RTL Compiler (RC) Version
Generated on:          Date
Module:                test
Technology library:   tutorial 1.0
Operating conditions: typical_case (balanced_tree)
Wirereload mode:       enclose
=====

Clock Description
-----
Clock
Name    Period   Rise     Fall      Domain   Pin/Port   No of Registers
-----
```

Name	Period	Rise	Fall	Domain	Pin/Port	No of Registers
CLK1	4000.0	0.0	2000.0	domain_1	Clk	5
CLK2	2000.0	0.0	1000.0	domain_1	C	0
CLK3	3000.0	0.0	1500.0	domain_2	Clk1	5

Command Reference for Encounter RTL Compiler

Analysis and Report

```
CLK4      6000.0  0.0   3000.0  domain_2    C1          0
Clock Network Latency / Setup Uncertainty
-----
Clock      Network Latency   Network Latency   Source Latency   Source Latency   Setup Uncertainty   Setup Uncertainty
Name       Rise             Fall            Rise           Fall          Rise           Fall
-----
CLK1      140.0           140.0          150.0         150.0        100.0          100.0
CLK2      120.0           120.0          120.0         120.0        110.0          110.0
CLK3      100.0           100.0          100.0         100.0        100.0          100.0
CLK4      0.0              0.0            0.0           0.0          0.0            0.0
Clock Relationship (with uncertainty & latency)
-----
From     To      R->R    R->F    F->R    F->F
-----
CLK1    CLK1    3900.0  1900.0  1900.0  3900.0
CLK1    CLK2    1840.0  840.0   1840.0  840.0
CLK2    CLK1    1950.0  1950.0  950.0   950.0
CLK2    CLK2    1890.0  890.0   890.0   1890.0
CLK3    CLK3    2900.0  1400.0  1400.0  2900.0
CLK3    CLK4    2800.0  2800.0  1300.0  1300.0
CLK4    CLK3    3100.0  1600.0  3100.0  1600.0
CLK4    CLK4    6000.0  3000.0  3000.0  6000.0
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

[Reporting Clocks](#)

[Generating Clock Reports](#)

[Analyzing and Reporting Multi-Mode Information](#)

Affected by this command: [create_mode](#) on page 287

Command Reference for Encounter RTL Compiler

Analysis and Report

report congestion

```
report congestion [ >file]
```

Reports the total number (and percentage) of gcells with overflow, the total overflow of the design as well as the maximum overflow and the associated gcell.

Options and Arguments

file Specifies the name of the file to which to write the report.

Example

The following command shows the congestion summary for design test.

```
rc:/> report congestion
```

```
=====  
Generated by:          RTL Compiler (RC) Version  
Generated on:         Date  
Module:               test  
Technology libraries: lib1  
                      lib2  
Operating conditions: max  
Interconnect mode:   placement  
Area mode:            physical library  
=====
```

GCELLS with H overflow: 99 (6.71%)
GCELLS with V overflow: 329 (22.29%)
Total number of GCELLS: 1476

Item	Oflow/Avail	Gcell	(Bounding-box location)
Max. Overflow	-111/201	20,18	(480.0,432.0), (504.0,456.0)
Max. Overflow (H)	-52/118	31,29	(744.0,696.0), (768.0,720.0)
Max. Overflow (V)	-75/87	19,17	(456.0,408.0), (480.0,432.0)
Total Overflow	-7172		
Total Overflow (H)	-1329		
Total Overflow (V)	-5843		

Related Information

Affected by this command: **synthesize -to placed**

report datapath

```
report datapath [-full_path]
    [-no_header] [-no_area_statistics]
    [-mux] [-all] [-max_width string]
    [-print_instantiated] [-print_inferred]
    [-sort keys] [design] [> file]
```

Reports datapath operators that were inferred from the design. This command is available after elaboration. You must set the `hdl_track_filename_row_col` attribute to true to enable filename, column, and line number tracking in the datapath report; otherwise these headings will be hidden.

Options and Arguments

<code>-all</code>	Reports all datapath operators present in the design including muxes. Note: The mux operators are different from the MUX library cells that are picked by the mapper or are available in the technology library.
<code>design</code>	Specifies a particular design on which to report datapath operators. By default, RTL Compiler reports on the current design.
<code>file</code>	Specifies the name of the file to which to write the report.
<code>-full_path</code>	Reports the full UNIX path name of the filename, including the filename. By default, RTL Compiler only reports the design name. See Examples for more information.
<code>-max_width <i>string</i></code>	Specifies the maximum width of individual column names. By default, the maximum width for a column is 20. If a name is more than 20, it will wrap to the next line. The valid column names are <i>Operator</i> , <i>Signedness</i> , <i>Inputs</i> , <i>Outputs</i> , <i>Cell Area</i> , <i>Line</i> , <i>Col</i> , and <i>Filename</i> .
<code>-mux</code>	Reports muxes present in the design. Muxes are not reported by default. Using the <code>-mux</code> option only displays the muxes in the design and suppresses the other datapath operators. To view both, use the <code>-all</code> option.

Command Reference for Encounter RTL Compiler

Analysis and Report

<code>-no_area_statistics</code>	Suppresses the table that only shows the total area and percentage information. The area and the percentage of the total area consumed by the datapath operators in the design are only available after issuing the <code>synthesize -to_mapped</code> command.
<code>-no_header</code>	Suppresses the header information.
<code>-print_inferred</code>	Reports only the inferred datapath components in the design.
<code>-print_instantiated</code>	Reports only the instantiated datapath components in the design.
<code>-sort keys</code>	Indicates how to sort the report. You can sort on the following keys: <ul style="list-style-type: none">■ <code>architectures</code> sorts the architectures in alphabetical order■ <code>area</code> sorts by descending area■ <code>inputs</code> sorts based on the number*width (number of bits) of the input signals—components with higher number of bits are printed first■ <code>instances</code> sorts the instances in alphabetical order■ <code>outputs</code> sorts based on the number*width (number of bits) of the output signals—components with higher number of bits are printed first■ <code>operators</code> sorts by operator■ <code>slack</code> sorts by ascending slack■ <code>subdesigns</code> sorts the subdesigns in alphabetical order By default, the report does not contain the slack numbers. Note: You can sort on several keys.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example generates a report of the datapath components for the rpdpl_basic design.

```
rc:/> report datapath rpdpl_basic
=====
Module:          rpdpl_basic
Technology library: tutorial 1.0
...
=====
Instantiated datapath components

      Operator Signedness  Inputs Outputs CellArea Line Col  Filename
=====
rpdpl_basic
  d1
    u2
      module:CW_CW_multadd_builtin_wA8_wB8_wC8_wZ16
        CW/CW_multadd/builtin
          n/a   n/a       8x8x8x1 16      1239.75  38  52 impl_inf.v
+++++
  mul_1_19
    very_fast/non_booth
      *      signed      9x9     16      912.75
+++++
  add_1_24
    very_fast      +      signed     16x9     16      322.50
=====

Inferred components

      Operator Signedness  Inputs Outputs CellArea Line Col  Filename
=====
rpdpl_basic
  mul_I8_28
    module:mult_unsigned
      very_fast/non_booth
        *      unsigned    16x8     16      1044.00  18  28 impl_inf.v
=====

      Type      CellArea Percentage
-----
datapath modules  2283.75      20.55
external muxes    0.00        0.00
others            8829.00     79.45
-----
total             11112.75     100.00
```

- By default, when using the `report datapath` command on a mapped netlist containing datapath operators, you will get the area statistics of the design, as shown in the following example:

```
-----
datapath modules 4938.00      100.00
mux modules      0.00        0.00
others           0.00        0.00
-----
total            4938.00      100.00
```

Command Reference for Encounter RTL Compiler

Analysis and Report

This information is useful in determining the percentage of the design that contains datapath operators. If you do not want to report this information, then use the `-no_area_statistics` option.

By default, the area report is suppressed for a netlist that contains only generic cells (no library cells).

- The following command provides a 30-character width to the `filename` column and provides a 0-character width to the `area` column:

```
report datapath -max_width {{filename 30} {area 0}}
```

- The following command generates a report sorted by area.

```
rc:/> report datapath -sort area
=====
...
=====
```

Inferred components

Operator	Signedness	Inputs	Outputs	CellArea	Line	Col	Filename
lt_leq							
add_14_26							
module:add_signed							
very_fast	+	signed	4x4	4			31.50 14 26 lt_leq.v
lt_leq							
lt_13_25							
module:lt_signed							
very_fast	<	signed	4x4	1			26.25 13 25 lt_leq.v
lt_leq							
lt_16_25							
module:lt_unsigned							
very_fast	<	unsigned	4x4	1			26.25 16 25 lt_leq.v
lt_leq							
le_17_26							
module:leq_unsigned							
very_fast	<=	unsigned	4x4	1			25.50 17 26 lt_leq.v

Type		CellArea	Percentage				
datapath modules		109.50	75.26				
external muxes		0.00	0.00				
others		36.00	24.74				
total		145.50	100.00				

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command sorts the report first by slack, then by area:

```
rc:/> report datapath -sort {slack area}
=====
...
=====

Inferred components

          Operator Signedness Inputs Outputs CellArea Line Col Filename Slack
=====
=====
lt_leq
 lt_13_25
 module:lt_signed
 very_fast < signed     4x4      1           26.25   13 25 lt_leq.v -145.2
=====
lt_leq
 lt_16_25
 module:lt_unsigned
 very_fast < unsigned    4x4      1           26.25   16 25 lt_leq.v -145.2
=====
lt_leq
 le_17_26
 module:leq_unsigned
 very_fast <= unsigned    4x4      1           25.50   17 26 lt_leq.v -53.3
=====
lt_leq
 add_14_26
 module:add_signed
 very_fast + signed     4x4      4           31.50   14 26 lt_leq.v 49.0
=====

          Type      CellArea Percentage
-----
datapath modules    109.50      75.26
external muxes      0.00       0.00
others              36.00      24.74
-----
total                145.50     100.00
```

Related Information

[Datapath Reporting in Datapath Synthesis in Encounter RTL Compiler](#)

Affected by this attribute:

[hdl_track_filename_row_col](#)

report design_rules

```
report design_rules [design]... [> file]
```

Reports any design rule violations that are present in the specified design objects.

Options and Arguments

<i>design</i>	Specifies the design for which you want to generate a report. By default, a report is created for all designs currently loaded in memory.
<i>file</i>	Specifies the name of the file to which to write the report.

Examples

- The following example generates a report of the design rule violations for the specified design:

```
> report design_rules alu
=====
Generated by:      RTL Compiler (RC) version
Generated on:      Current date Current time
Module:           alu
Technology library: tutorial 1.0
Operating conditions: typical_case (balanced_tree)
Wireload mode:    enclosed
=====
Max_transition design rule: no violations.

Max_capacitance design rule: no constraints.

Max_fanout design rule: no constraints.
```

Related Information

[Setting Design Rule Constraints](#) in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

Affected by this command: [synthesize](#) on page 348

Affected by these attributes: [ignore_library_max_fanout](#)
[max_capacitance](#)
[max_fanout](#)
[max_transition](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report dft_chains

```
report dft_chains [design] [-chain scan_chain]...
    [-dft_configuration_mode dft_config_mode_name]
    [-opcg_side_scan]
    [-summary] [> file]
```

Reports for every chain in the design the following elements:

- The scan data input port and its hookup pin
 - The scan data output port and its hookup pin
 - The shift enable port and its hookup pin
 - The DFT clock domain and the DFT clock domain edge of the chain (for muxed scan only)
 - The length of the chain
 - The elements in the chain

In case of the muxed scan style, the report also lists for each element the test clock and test clock edge determined by the `check_dft_rules` command for that element.

- Any user-specified segments with their elements
 - In case of the muxed scan style, this includes data lockup elements, and the location of the data lockup element in the chain.
 - The bit position and length of any user-specified segment in the chain
 - The head and tail test clock and test clock edge for any abstract segment in the chain

Options and Arguments

-chain *scan_chain*...

Reports only the listed scan chains

design

Specifies the design for which you want to generate a report.

Note: If you have multiple top designs, you must either specify the design name, or change to the directory in the design hierarchy that contains the design for which you want the report.

`-dft_configuration_mode dft_configuration_mode_name`

Reports the scan chains related to the specified scan mode.

Command Reference for Encounter RTL Compiler

Analysis and Report

<code>file</code>	Specifies the name of the file to which to write the report.
<code>-opcg_scan_side</code>	Reports only the side scan chains that contain the domain macro segments.
<code>-summary</code>	Condenses the scan chain report to include only chain name, scan-data pins, shift enable, clock domain and length information.

Examples

- The following example shows one scan chain, with three user-defined segments.

```
rc:/designs/test> report dft_chains
Reporting 1 scan chain

Chain 1: DFT_chain_1
  scan_in:      in[0]
  scan_out:     out[1] (shared output)
  shift_enable: SE (active high)
  clock_domain: clk (edge: mixed)
  length: 8
    START segment segHead (type: floating)
      # @ bit 1, length: 2
      bit 1      out_reg_4 <clk/fall>
      bit 2      out_reg_5 <clk/fall>
    END segment segHead
    bit 3      out_reg_6 <clk/fall>
    bit 4      out_reg_7 <clk/fall>
    START segment segBody (type: fixed)
      # @ bit 5, length: 2
      bit 5      out_reg_1 <clk/rise>
      bit 6      out_reg_3 <clk/rise>
    END segment segBody
    bit 7      out_reg_2
    START segment segTail (type: floating)
      # @ bit 8, length: 1
      bit 8      out_reg_0 <clk/rise>
    END segment segTail
-----
```

- The following example shows the summary report for the previous example.

```
rc:/designs/test> report dft_chains -summary
Reporting 1 scan chain (muxed_scan)

Chain 1: DFT_chain_1
  scan_in:      in[0]
  scan_out:     out[1] (shared output)
  shift_enable: SE (active high)
  clock_domain: clk (edge: mixed)
  length: 8
-----
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following examples show the scan chains of a design before and after DFT compression:

- **Before compression**

```
Reporting 2 scan chains (muxed_scan)
```

```
Chain 1: AutoChain_1
  scan_in:      DFT_sdi_1
  scan_out:     DFT_sdo_1
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
    bit 1        out_reg[1]  <clk (rise)>
    bit 2        out_reg[2]  <clk (rise)>
    bit 3        out_reg[3]  <clk (rise)>
    bit 4        out_reg[4]  <clk (rise)>
    bit 5        out_reg[5]  <clk (rise)>
    bit 6        out_reg[6]  <clk (rise)>
    bit 7        out_reg[7]  <clk (rise)>
    bit 8        out_reg[8]  <clk (rise)>
    bit 9        out_reg[9]  <clk (rise)>
    bit 10       out_reg[10] <clk (rise)>
-----
Chain 2: AutoChain_2
  scan_in:      DFT_sdi_2
  scan_out:     DFT_sdo_2
  shift_enable: se (active high)
  clock_domain: clk (edge: rise)
  length: 10
    bit 1        out_reg[11] <clk (rise)>
    bit 2        out_reg[12] <clk (rise)>
    bit 3        out_reg[13] <clk (rise)>
    bit 4        out_reg[14] <clk (rise)>
    bit 5        out_reg[15] <clk (rise)>
    bit 6        out_reg[16] <clk (rise)>
    bit 7        out_reg[17] <clk (rise)>
    bit 8        out_reg[18] <clk (rise)>
    bit 9        out_reg[19] <clk (rise)>
    bit 10       out_reg[20] <clk (rise)>
```

- **After compression**

The report shows in addition

- How the original scan chains have been divided in several internal chains.
- For each internal chain the START and END (separate scan data input and output) are given together with the length of the internal channel.
- An additional scan chain, `mask_chain` (if the user opted to add making logic)

The mask chain is comprised of abstract segments only.

The shift-enable signal of the mask chain is reported as NONE because it is a *connected* shift enable.

Command Reference for Encounter RTL Compiler

Analysis and Report

```
Reporting 3 scan chains (muxed_scan)

Chain 1: AutoChain_1 (compressed)
scan_in: DFT_sdi_1
scan_out: DFT_sdo_1
shift_enable: se (active high)
clock_domain: clk (edge: rise)
length: 10
    <START compressed internal chain AutoChain_1_0 (sdi: g121/SWBOX_SI[0])>
    bit 1      out_reg[1]  <clk (rise)>
    bit 2      out_reg[2]  <clk (rise)>
    <END   compressed internal chain AutoChain_1_0 (sdo: g121/SWBOX_SO[0])
(length: 2)>
    <START compressed internal chain AutoChain_1_1 (sdi: g121/SWBOX_SI[1])>
    bit 3      out_reg[3]  <clk (rise)>
    bit 4      out_reg[4]  <clk (rise)>
    <END   compressed internal chain AutoChain_1_1 (sdo: g121/
SWBOX_SO[1]) (length: 2)>
    <START compressed internal chain AutoChain_1_2 (sdi: g121/SWBOX_SI[2])>
    bit 5      out_reg[5]  <clk (rise)>
    bit 6      out_reg[6]  <clk (rise)>
    <END   compressed internal chain AutoChain_1_2 (sdo: g121/SWBOX_SO[2])
(length: 2)>
    <START compressed internal chain AutoChain_1_3 (sdi: g121/SWBOX_SI[3])>
    bit 7      out_reg[7]  <clk (rise)>
    bit 8      out_reg[8]  <clk (rise)>
    <END   compressed internal chain AutoChain_1_3 (sdo: g121/SWBOX_SO[3])
(length: 2)>
    <START compressed internal chain AutoChain_1_4 (sdi: g121/SWBOX_SI[4])>
    bit 9      out_reg[9]  <clk (rise)>
    bit 10     out_reg[10] <clk (rise)>
    <END   compressed internal chain AutoChain_1_4 (sdo: g121/SWBOX_SO[4])
(length: 2)>
-----
Chain 2: AutoChain_2 (compressed)
scan_in: DFT_sdi_2
scan_out: DFT_sdo_2
shift_enable: se (active high)
clock_domain: clk (edge: rise)
length: 10
    <START compressed internal chain AutoChain_2_5 (sdi: g121/SWBOX_SI[5])>
    bit 1      out_reg[11] <clk (rise)>
    bit 2      out_reg[12] <clk (rise)>
    <END   compressed internal chain AutoChain_2_5 (sdo: g121/SWBOX_SO[5])
(length: 2)>
...
...
    <START compressed internal chain AutoChain_2_9 (sdi: g121/SWBOX_SI[9])>
    bit 9      out_reg[19] <clk (rise)>
    bit 10     out_reg[20] <clk (rise)>
    <END   compressed internal chain AutoChain_2_9 (sdo: g121/SWBOX_SO[9])
(length: 2)>
-----
Warning - could not find shift_enable signal for chain /designs/top/dft/
report/actual_scan_chains/mask_chain
Chain 3: mask_chain
scan_in: msi
scan_out: mso
shift_enable: NONE
clock_domain: mck (edge: rise)
length: 10
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
START segment DFT_segment_1 (type: abstract)
  # @ bit 1, length: 4
  pin      g121/msi[0] <mck (rise)>
  pin      g121/mso[0] <mck (rise)>
END segment DFT_segment_1
START segment DFT_segment_3 (type: abstract)
  # @ bit 5, length: 3
  pin      g121/msi[2] <mck (rise)>
  pin      g121/mso[2] <mck (rise)>
END segment DFT_segment_3
START segment DFT_segment_2 (type: abstract)
  # @ bit 8, length: 3
  pin      g121/msi[1] <mck (rise)>
  pin      g121/mso[1] <mck (rise)>
END segment DFT_segment_2
-----
```

- The following command reports two side scan chains, each with one OPCG segment.

```
rc:/> report dft_chains -opcg_side_scan

Reporting 2 side scan chains
Side-scan chain 1:
scan_in: io[4] (hookup: p/p4/Y)
clock domain: opcg_load_clk
length: 7
  START segment OPCG_DOMAIN_SEG_1 (type: abstract)
    #@bit 1, length 7
    pin      DFT_OD2/PGMSI
    pin      DFT_OD2/PGMSO
  END segment OPCG_DOMAIN_SEG_1
-----
Side-scan chain 2:
scan_in: DFT_sdi_1
clock domain: opcg_load_clk
length: 6
  START segment OPCG_DOMAIN_SEG_0 (type: abstract)
    #@bit 1, length 6
    pin      DFT_OD1/PGMSI
    pin      DFT_OD1/PGMSO
  END segment OPCG_DOMAIN_SEG_0
-----
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Reporting on All Scan Chains](#)
- [Compressing Scan Chains](#)
- [Concatenating Scan Chains](#)

Affected by this command:

[connect scan chains](#) on page 620

[compress scan chains](#) on page 601

Related attributes:

[Actual Scan Chain](#) attributes

[Actual Scan Segment](#) attributes

report dft_core_wrapper

```
report dft_core_wrapper
  [-wrapped | -unwrapped]
  [-inside_core] [-location {pin|port|subport}...]
  [-report_flops] [-summary]
  [-max_print_objects integer]
  [design|instance] [>file]
```

Reports the wrapper cells inserted in the design.

Options and Arguments

{ <i>design</i> <i>instance</i> }	Specifies the design or instance for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
<i>file</i>	Specifies the name of the file to which to write the report.
-inside_core	Reports the wrapper cells inside the core instance.
-location	Limits the reported wrapper cells to the specified pin(s), port(s) or subport(s).
-max_print_objects <i>integer</i>	Specifies the maximum number of wrapper segments or flops to be reported on any pin, port or subport. <i>Default:</i> 5
-report_flops	Reports the flops instead of the wrapper segments.
-summary	Reports the summary only.
-unwrapped	Lists only those ports or pins that do not have a wrapper cell associated with them.
-wrapped	Lists only those ports or pins that have an associated wrapper cell.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following examples illustrate the different report options on the same design.

```
rc:/> report dft_core_wrapper test
Reporting 13 ports
Port/Pin          type      usage      wrapper_objects
=====  ======  =====
WEXT             wrapper_control -          -
WIG              wrapper_control -          -
WINT             wrapper_control -          -
WOG              wrapper_control -          -
WSEN_in          wrapper_control -          -
clk              wrapper_control -          -
WSEN_out         shift_enable   -          -
se               shift_enable   -          -
in1              shared       input       wrap_1
in1              shared       output      wrap_4
in2              shared       input       wrap_3
in2              shared       output      wrap_1
in3              shared       input       wrap_4
out1             shared       input       wrap_3
out1             shared       output      wrap_2
out1             shared       input       wrap_1
out2             shared       input       wrap_4
out2             shared       output      wrap_1
out2             shared       input       wrap_4

Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0      input bounding :0      output bounding :0
Number of Shared wrapper cells :4        input bounding :3      output bounding :1
-----
Total no. of wrapper cells      :4      input bounding :3      output bounding :1
```

```
rc:/> report dft_core_wrapper test -wrapped
```

```
Reporting 5 ports
Port/Pin          type      usage      wrapper_objects
=====  ======  =====
in1              shared       input       wrap_1
in1              shared       output      wrap_4
in2              shared       input       wrap_3
in2              shared       output      wrap_1
in2              shared       input       wrap_4
in3              shared       input       wrap_3
out1             shared       input       wrap_2
out1             shared       output      wrap_1
out1             shared       input       wrap_4
out2             shared       input       wrap_1
out2             shared       output      wrap_4

Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0      input bounding :0      output bounding :0
Number of Shared wrapper cells :4        input bounding :3      output bounding :1
-----
Total no. of wrapper cells      :4      input bounding :3      output bounding :1
```

```
rc:/> report dft_core_wrapper -report_flops
```

```
Reporting 13 ports
Port/Pin          type      usage      wrapper_objects
=====  ======  =====
WEXT             wrapper_control -          -
WIG              wrapper_control -          -
WINT             wrapper_control -          -
WOG              wrapper_control -          -
WSEN_in          wrapper_control -          -
clk              wrapper_control -          -
WSEN_out         shift_enable   -          -
se               shift_enable   -          -
in1              shared       input       tmp5_reg
in1              shared       output      tmp4_reg
in1              shared       output      tmp3_reg
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```

in2           shared      input      tmp5_reg
in2           shared      output     tmp4_reg
in3           shared      input      tmp3_reg
out1          shared      input      tmp2_reg
out2          shared      input      tmp5_reg
                  shared      input      tmp4_reg
                  shared      input      tmp5_reg
                  shared      input      tmp4_reg

Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0      input bounding :0      output bounding :0
Number of Shared wrapper cells   :4      input bounding :3      output bounding :1
-----
Total no. of wrapper cells       :4      input bounding :3      output bounding :1

rc:/> report dft_core_wrapper test -summary
Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0      input bounding :0      output bounding :0
Number of Shared wrapper cells   :4      input bounding :3      output bounding :1
-----
Total no. of wrapper cells       :4      input bounding :3      output bounding :1

```

- The following reports shows the wrapper cells inside the core instance `i_core1`. In the first report, wrapper cells were inserted on the input list. In the second report, wrapper cells were also inserted on the output list.

```
rc:/> report dft_core_wrapper -inside_core i_core1
```

first report:

```

Reporting 15 ports/pins
Port/Pin          type      usage      wrapper_objects
=====            ===      =====      =====
i_core1/se        wrapper_control    -
i_core1/wclk      wrapper_control    -
i_core1/wext      wrapper_control    -
i_core1/wint      wrapper_control    -
i_core1/tclk      test_clock        -
i_core1/tm        test_control      -
i_core1/clk       excluded         -
i_core1/out[0]    not processed    -
i_core1/out[1]    not processed    -
i_core1/out[2]    not processed    -
i_core1/out[3]    not processed    -
i_core1/in[0]     shared          input      wrap_in_3
i_core1/in[1]     shared          input      wrap_in_4
i_core1/in[2]     shared          input      wrap_in_1
i_core1/in[3]     shared          input      wrap_in_2

```

```

Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0      input bounding :0      output bounding :0
Number of Shared wrapper cells   :4      input bounding :4      output bounding :0
-----
Total no. of wrapper cells       :4      input bounding :4      output bounding :0

```

second report:

```

Reporting 15 ports/pins
Port/Pin          type      usage      wrapper_objects
=====            ===      =====      =====
i_core1/se        wrapper_control    -
i_core1/wclk      wrapper_control    -
i_core1/wext      wrapper_control    -
i_core1/wint      wrapper_control    -
i_core1/tclk      test_clock        -
i_core1/tm        test_control      -
i_core1/clk       excluded         -
i_core1/in[0]     shared          input      wrap_in_3
i_core1/in[1]     shared          input      wrap_in_4
i_core1/in[2]     shared          input      wrap_in_1

```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
i_core1/in[3]           shared      input      wrap_in_2
i_core1/out[0]          dedicated   output     wrap_out_4
i_core1/out[1]          dedicated   output     wrap_out_3
i_core1/out[2]          dedicated   output     wrap_out_2
i_core1/out[3]          dedicated   output     wrap_out_1

Wrapper insertion summary report
=====
Number of Dedicated wrapper cells :0    input bounding :0    output bounding :0
Number of Shared wrapper cells  :4    input bounding :4    output bounding :0
-----
Total no. of wrapper cells       :4    input bounding :4    output bounding :0
```

Related Information

Affected by this command: [insert_dft_wrapper_cell](#) on page 775

report dft_registers

```
report dft_registers [-pass_tdrc] [-fail_tdrc]
    [-lockup] [-latch] [-dont_scan] [-misc]
    [-shift_reg] [-multi-bit] [design] [> file]
```

Reports the DFT status of all flip-flop instances in the design. Use this command after running `check_dft_rules`. More specifically, the command reports

- Registers which pass the DFT rule checker
 - For each flip-flop, it reports the hierarchical instance name along with their DFT test clock domain and active edge.
- Registers which fail the DFT rule checker
 - For each flip-flop, it reports the hierarchical instance name along with the violation type (clock, or asynchronous set/reset) and the violation Id number.
 - For an abstract segment that fails the DFT rule checker, it reports the name of the abstract segment, and the number of flip-flops in the segment.
- Registers that are preserved or marked dont - scan
 - Note:** Mapped flip-flops can be listed in this category if
 - The flip-flop is marked preserved and it is mapped to a non-scan flop
 - However, if a flip-flop is marked preserved and is already mapped to scan for DFT purposes, it will be listed as either passing or failing the DFT rule checker—based on the DFT rule checker analysis—and it will not be listed in this category.
 - The flip-flop is mapped to a scan flop for non-DFT purposes
 - Registers that are marked Abstract Segment dont-scan.
 - Registers that are part of shift register segments
 - Registers that are identified as lockup elements
 - Registers that are level-sensitive elements
 - Registers that are implemented as multibit
 - Registers not part of any of the other categories

Without any options specified, the command reports on all categories of registers.

Command Reference for Encounter RTL Compiler

Analysis and Report

Options and Arguments

<i>design</i>	Specifies the design for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
-dont_scan	Reports all edge-triggered registers that are not to be mapped to scan flops, or connected into the top-level chains.
-fail_tdrc	Reports all edge-triggered registers that fail the DFT rule checks.
<i>file</i>	Specifies the name of the file to which to write the report.
-latch	Reports all registers of type latch in the design. Note: Latch instances which are instantiated as lockup elements in a preserved segment are reported with the -lockup option.
-lockup	Reports data lockup elements of type latch or flop that are either instantiated in a preserved segment, or added to the design during scan chain configuration.
-misc	Reports all registers that are not reported through any of the other options, such as clock-gating registers.
-multi_bit	Reports all registers that are implemented as multibit elements.
-pass_tdrc	Reports all edge-triggered registers that pass the DFT rule checks.
-shift_reg	Reports all registers that are part of shift register segments.

Example

- The following example shows that 1 flip-flop passed the DFT rule checks, while 4 flip-flops failed the tests.

```
rc:/> report dft_registers
Reporting registers that pass DFT rules
  Iset reg      PASS; Test clock: clk/rise
Reporting registers that fail DFT rules
  out_reg_0    FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_1    FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_2    FAIL; violations: clock #(0 ) async set #(1 )
  out_reg_3    FAIL; violations: clock #(0 ) async set #(1 )
Reporting registers that are preserved or marked dont-scan
Reporting registers that are marked Abstract Segment Dont Scan
Reporting registers that are part of shift register segments
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
Reporting registers that are identified as lockup elements
Reporting registers that are level-sensitive elements
Reporting misc. non-scan registers
Summary:
Total registers that pass DFT rules: 1
Total registers that fail DFT rules: 4
Total registers that are marked preserved or dont-scan: 0
Total registers that are marked Abstract Segment dont-scan: 0
Total registers that are part of shift register segments: 0
Total registers that are lockup elements: 0
Total registers that are level-sensitive: 0
Total registers that are misc. non-scan: 0
```

- The following report was generated after the scan configuration engine was run. In this case, four lockup elements were inserted:

```
rc:/> report dft_registers

Reporting registers that pass DFT rules
out1_reg_0      PASS; Test clock: test_clk1/rise
...
out1_reg_4      PASS; Test clock: test_clk1/fall
...
out2_reg_0      PASS; Test clock: test_clk2/rise
...
out2_reg_4      PASS; Test clock: test_clk2/fall
...
out3_reg_0      PASS; Test clock: test_clk3/rise
...
out3_reg_4      PASS; Test clock: test_clk3/fall
...
Reporting registers that fail DFT rules
Reporting registers that are preserved or marked dont-scan
Reporting registers that are marked Abstract Segment Dont Scan
Reporting registers that are part of shift register segments
Reporting registers that are identified as lockup elements
DFT_lockup_g1
DFT_lockup_g348
DFT_lockup_g349
DFT_lockup_g350
Reporting registers that are level-sensitive elements
Reporting misc. non-scan registers
Summary:
Total registers that pass DFT rules: 27
Total registers that fail DFT rules: 0
Total registers that are marked preserved or dont-scan: 0
Total registers that are marked Abstract Segment dont-scan: 0
Total registers that are part of shift register segments: 0
Total registers that are lockup elements: 4
Total registers that are level-sensitive: 0
Total registers that are misc. non-scan: 0
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Reporting Status on All Flip-Flops](#)
- [Controlling Mapping to Scan Flip-Flops](#)

Affected by this command: [check_dft_rules](#) on page 589

Related attributes: [dft_dont_scan](#)

[dft_scan_style](#)

[dft_status](#)

[dftViolation](#)

[preserve](#) (instance)

[preserve](#) (subdesign)

[Scan Segment Attributes](#)

report dft_setup

```
report dft_setup [design] [> file]
```

Reports DFT setup information for the design including both user-defined and tool-inferred information.

You must load and elaborate a design before you can use this command. The contents of this report further depends on the stage of the design that you are at.

Use this command at the end of the design process to document the DFT setup and resulting configuration of the design.

Options and Arguments

design Specifies the design for which you want to generate a report.

By default a report is created for all designs currently loaded in memory.

file Specifies the name of the file to which to write the report.

Examples

- The following example shows the report after you have elaborated the design. Because you have not specified any setup information yet, the information shown is based on the default settings for DFT.

```
rc:/> report dft_setup

Design Name
=====
top

Scan Style
=====
 muxed_scan
DFT rule check status is not available. Need to (re)run check_dft_rules

Global Constraints
=====
 Minimum number of scan chains: no_value
 Maximum length of scan chains: no_value
 Lock-up element type: level_sensitive
 Mix clock edges in scan chain: false
 Prefix for unnamed scan objects: DFT_

Test signal objects
=====

Test clock objects
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
=====
Reporting all test clocks as TDRC status is not available

DFT controllable objects
=====

DFT don't scan objects
=====

DFT abstract don't scan objects
=====

DFT scan segment constraints
=====

DFT scan chain constraints
=====

DFT actual scan chains
=====
```

- The following example shows the report after the scan configuration engine was run:

```
rc:/designs/test> report dft_setup

Design Name
=====
    top

Scan Style
=====
    muxed_scan
Design has a valid DFT rule check status

Global Constraints
=====
    Minimum no of Scan chains: no_value
    Maximum length of scan chains: no_value
    Lock-up element type: level_sensitive
    Mix clock edges in scan chain: true
    Prefix to name user defined scan chains: DFT_

Test signal objects
=====
    shift_enable:
        object name: SE
        pin name: SE
        hookup_pin: SE
        hookup_polarity: non_inverted
        active: high
        ideal: true
        user defined: true

Test clock objects
=====
    test_clock:
        object name: clk
        user defined: false
        source: clk
        root source: clk
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
root source polarity: non_inverting
hookup_pin: clk
period: 50000.0

DFT controllable objects
=====
DFT don't scan objects
=====
DFT abstract don't scan objects
=====
DFT scan segment constraints
=====
Segment:
    object name: segHead
    scan-in:
    scan-out:
    shift-enable: internal
    connected_shift_enable: false
    length: 2
    type: floating

Segment:
    object name: segTail
    ...
    ...

Segment:
    object name: segBody
    scan-in:
    scan-out:
    shift-enable: internal
    connected_shift_enable: false
    length: 2
    type: fixed

DFT scan chain constraints
=====
User Chain:
    object name: DFT_chain_1
    scan-in: in[0]
    scan-in hookup_pin: in[0]
    scan-out: out[1]
    scan-out hookup_pin: out[1]
    shared out: true
    shift_enable object name:
    max length: no_value
    head segment: segHead
    tail segment: segTail
    complete: false

DFT actual scan chains
=====
Actual Chain:
    object name: DFT_chain_1
    scan-in: in[0]
    scan-in hookup_pin: in[0]
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
scan-out: out[1]
scan-out hookup_pin: out[1]
shared out: true
shift_enable: SE
length: 8
segment objects: segHead segBody segTail
analyzed: false
test_clock domain: clk
test_clock edge: any
```

Related Information

Recommended Flow in Design for Test in Encounter RTL Compiler

Affected by these constraints: [define_dft_shift_enable](#) on page 686

[define_dft_test_clock](#) on page 693

[define_dft_test_mode](#) on page 697

Affected by this command: [check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

Related attributes: [dft_controllable](#)

[dft_dont_scan](#)

[dft_lockup_element_type](#)

[dft_max_length_of_scan_chains](#)

[dft_min_number_of_scan_chains](#)

[dft_mix_clock_edges_in_scan_chains](#)

[dft_prefix](#)

[dft_scan_style](#)

(instance) [preserve](#)

(subdesign) [preserve](#)

Related attributes: [Actual Scan Chain Attributes](#)

[Actual Scan Segment Attributes](#)

[Scan Segment Attributes](#)

[Scan Chain Attributes](#)

[Test Clock Attributes](#)

[Test Signal Attributes](#)

report dft_violations

```
report dft_violations [-async] [-clock]
                      [-abstract] [-shiftreg] [-tristate]
                      [-race] [-xsource] [-xsource_by_instance]
                      [design] [> file]
```

Reports for each violation the object name, the type of violation, the description of the violation, how to find the root pin or port of the problem, the source file and the line number where to find the problem, the number of registers affected, and the instance names of the affected registers. The report is sorted by violation type. Run the DFT rule checker to get meaningful information.

Without any options specified, the command reports on all types of violations.

Options and Arguments

-abstract	Reports all abstract segment test mode violations.
-async	Reports all async set and async reset violations.
-clock	Reports all clock violations.
design	Specifies the design for which you want to generate a report. By default a report is created for all designs currently loaded in memory.
file	Specifies the name of the file to which to write the report.
-race	Specifies to report potential race condition violations.
-shiftreg	Reports all shift register segment violations.
-tristate	Reports tristate design rules checking violations.
-xsource	Reports X-Source violations.
-xsource_by_instance	Reports X-Source violations by instance.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example shows that the design has four violations, but only detailed information on the clock violations is requested.

```
rc:/> report dft_violations -clock
Total 4 violations (muxed_scan)

Clock Rule Violations
=====
Reporting 2 clock violations

Violation #0:
  Object name: vid_0_clock
  Type: clock violation
  Description: [CLOCK-05] internal or gated clock signal
  Source: g1/z (test10.v:12)
  Number of registers affected: 4
  Affected registers:
    out1_reg[0]
    out1_reg[1]
    out1_reg[2]
    out1_reg[3]

Violation #1:
  Object name: vid_1_clock
  Type: clock violation
  Description: [CLOCK-06] clock signal driven by a sequential element
  Source: divClk_reg/q (test10.v:14)
  Number of registers affected: 4
  Affected registers:
    out2_reg[0]
    out2_reg[1]
    out2_reg[2]
    out2_reg[3]

Violation Rule Summary Report
=====
[CLOCK-05] internal or gated clock signal : 1
[CLOCK-06] clock signal driven by a sequential element : 1
```

- The following example reports one tristate net contention violation.

```
rc:/> report dft_violations
Total 1 violation (muxed_scan)

Tristate Net Violations
=====
Reporting 1 tristate net contention violations

Violation #0:
  Object name: vid_0_tristate_net
  Type: tristate net violation
  Description: [TRISTATE_NET-01] tristate net potentially driven by conflicting
  values

  Tristate net affected: tbus
  Tristate net load pin affected: tbus
  Tristate drivers causing contention: b2/Y b1/Y b3/Y b4/Y b6/Y b8/Y b7/Y b5/Y
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
...
Violation Rule Summary Report
=====
[TRISTATE_NET-01] tristate net potentially driven by conflicting values : 1
```

Related Information

[Reporting the DFT Violations in Design for Test in Encounter RTL Compiler](#)

Affected by this command: [check_dft_rules](#) on page 589

[fix_dft_violations](#) on page 703

Related attributes: [Violations Attributes](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report_disabled_transparent_latches

```
report disabled_transparent_latches [> file]
```

Reports the disabled transparent latches and the enable to Q arcs that are not yet disabled. Transparent latches are latches with enable signal held constant at the active state. Without enabling transparent latches, paths through them cannot be traced.

Options and Arguments

file Specifies the name of the file to which to write the report.

Related Information

Affected by this command: [enable transparent latches](#) on page 73

Command Reference for Encounter RTL Compiler

Analysis and Report

report gates

```
report gates [-library_domain library_domain_list]
              [-power] [-yield]
              [-instance_hier instance] [design] [> file]
```

Reports the technology library cells that were implemented (and identifies their originating libraries), the area of the cell instances, and the break up of the instances into timing models, sequential cells, integrated clock-gating cells, inverters, buffers, and logic gate cells. Optionally power information can be reported.

Note: Timing models can refer to memory cells, IPs, and so on.

Options and Arguments

design Specifies the block for which you want to generate a report. You can also `cd` into the particular design directory and generate the report.

file Specifies the name of the file to which to write the report.

-instance hier *instance*

Restricts the reported information to the specified hierarchical instance.

-library domain *library domain list*

Restricts the reported information to the specified library domains.

Note: This option only applies when using CPF.

-power

Adds leakage power and internal power information

-yield

Reports the yield cost and yield percentage values for each library cell.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example reports information such as the type of cells that were used, number of instances, area, and originating library of the current design.

```
rc:/> report gates
=====
...
Module: alu
Technology library: tutorial 1.0
Operating conditions: typical_case (balanced_tree)
Wireload mode: enclosed
=====
Gate Instances Area Library
-----
flopdrs 9 72.000 tutorial
inv1 35 105.000 tutorial
nand2 112 112.000 tutorial
nor2 37 55.500 tutorial
xor2 17 34.000 tutorial
-----
total 210 378.500
-----
Type Instances Area Area %
-----
sequential 9 72.000 19.0
inverter 35 105.000 27.7
logic 166 201.500 53.2
-----
total 210 378.500 100.0
```

- The following example shows the additional power information in the report.

- The first table lists the leakage and internal power of all instances per used library cell.
 - The second table shows the number and percentage of instances coming from each library, the leakage and internal power consumed by these instances, as well as the percentage of power consumed by these instances.
 - The third table shows the leakage and internal power of all sequential cells, inverters, and combinational cells, as well as the percentage of power that each type consumes.

```
rc:/> report gates -power
=====
...
=====
Gate Instances Area Leakage Internal
Power (nW) Power (nW) Library
-----
ACHCONX2TH 1 28.856 59.948 372.198 slow_hvt
ADDHX1 1 39.040 158.364 276.327 slow
...
XOR2XL 10 118.820 339.664 471.216 slow
```

Command Reference for Encounter RTL Compiler

Analysis and Report

XOR2XLTH	449	5335.018	6259.161	22146.660	slow_hvt																																								
XOR3XL	1	28.856	83.655	80.446	slow																																								
total	11980	109000.238	139743.990	505460.121																																									
<table border="1"> <thead> <tr> <th>Library</th> <th>Instances</th> <th>Leakage Power %</th> <th>Leakage Power %</th> <th>Internal Power (nW)</th> <th>Internal Power %</th> </tr> </thead> <tbody> <tr> <td>slow</td> <td>1919</td> <td>16.0</td> <td>74607.161</td> <td>53.4</td> <td>168629.031</td> </tr> <tr> <td>slow_hvt</td> <td>10061</td> <td>84.0</td> <td>65136.830</td> <td>46.6</td> <td>336831.090</td> </tr> </tbody> </table>						Library	Instances	Leakage Power %	Leakage Power %	Internal Power (nW)	Internal Power %	slow	1919	16.0	74607.161	53.4	168629.031	slow_hvt	10061	84.0	65136.830	46.6	336831.090																						
Library	Instances	Leakage Power %	Leakage Power %	Internal Power (nW)	Internal Power %																																								
slow	1919	16.0	74607.161	53.4	168629.031																																								
slow_hvt	10061	84.0	65136.830	46.6	336831.090																																								
<table border="1"> <thead> <tr> <th>Type</th> <th>Instances</th> <th>Area</th> <th>Area %</th> <th>Leakage Power (nW)</th> <th>Leakage Power %</th> <th>Internal Power (nW)</th> <th>Internal Power %</th> </tr> </thead> <tbody> <tr> <td>sequential</td> <td>212</td> <td>7864.054</td> <td>7.2</td> <td>17285.068</td> <td>12.4</td> <td>112374.696</td> <td>22.2</td> </tr> <tr> <td>inverter</td> <td>1684</td> <td>5961.269</td> <td>5.5</td> <td>5752.468</td> <td>4.1</td> <td>22492.255</td> <td>4.4</td> </tr> <tr> <td>buffer</td> <td>37</td> <td>364.941</td> <td>0.3</td> <td>1877.474</td> <td>1.3</td> <td>3920.823</td> <td>0.8</td> </tr> <tr> <td>logic</td> <td>10047</td> <td>94809.9748</td> <td>87.0</td> <td>114828.979</td> <td>82.2</td> <td>366672.348</td> <td>72.5</td> </tr> </tbody> </table>						Type	Instances	Area	Area %	Leakage Power (nW)	Leakage Power %	Internal Power (nW)	Internal Power %	sequential	212	7864.054	7.2	17285.068	12.4	112374.696	22.2	inverter	1684	5961.269	5.5	5752.468	4.1	22492.255	4.4	buffer	37	364.941	0.3	1877.474	1.3	3920.823	0.8	logic	10047	94809.9748	87.0	114828.979	82.2	366672.348	72.5
Type	Instances	Area	Area %	Leakage Power (nW)	Leakage Power %	Internal Power (nW)	Internal Power %																																						
sequential	212	7864.054	7.2	17285.068	12.4	112374.696	22.2																																						
inverter	1684	5961.269	5.5	5752.468	4.1	22492.255	4.4																																						
buffer	37	364.941	0.3	1877.474	1.3	3920.823	0.8																																						
logic	10047	94809.9748	87.0	114828.979	82.2	366672.348	72.5																																						
total	11980	109000.2382	100.0	139743.990	100.0	505460.121	100.0																																						

- The following example reports the yield cost and yield percentage values for each library cell:

Gate	Instances	Area	Cost	Yield	Library																
flopdrs	33	264.000	3.39278e-06	99.9997	tutorial																
inv1	103	51.500	1.5022e-06	99.9998	tutorial																
nand2	315	315.000	1.08311e-05	99.9989	tutorial																
nor2	19	28.500	6.79989e-07	99.9999	tutorial																
total	470	659.000	1.64061e-05	99.9984																	
<table border="1"> <thead> <tr> <th>Type</th> <th>Instances</th> <th>Area</th> <th>Area %</th> </tr> </thead> <tbody> <tr> <td>sequential</td> <td>33</td> <td>264.000</td> <td>40.1</td> </tr> <tr> <td>inverter</td> <td>103</td> <td>51.500</td> <td>7.8</td> </tr> <tr> <td>logic</td> <td>334</td> <td>343.500</td> <td>52.1</td> </tr> </tbody> </table>						Type	Instances	Area	Area %	sequential	33	264.000	40.1	inverter	103	51.500	7.8	logic	334	343.500	52.1
Type	Instances	Area	Area %																		
sequential	33	264.000	40.1																		
inverter	103	51.500	7.8																		
logic	334	343.500	52.1																		
total	470	659.000	100.0																		

Related Information

[Generating a Gates Report in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

[Reporting Power Consumption per Used Cell Types in Low Power in Encounter RTL Compiler](#)

Affected by this command: [synthesize](#) on page 348

Command Reference for Encounter RTL Compiler

Analysis and Report

report hierarchy

```
report hierarchy  
    [-subdesign subdesign] [design] [> file]
```

Generates a report based on the design hierarchy starting from the top level module down to the leaf module. The report will take the following format:

```
level instance ( module ) <status>
status  : preserve_<value> -- indicating preserve hierarchy or inherited_preserve
value
      : blackbox -- indicating unresolved instance
```

Note: The `hdl_track_filename_row_col` attribute needs to be set to `true` before elaboration in order to successfully use this command.

Options and Arguments

design Specifies a specific design to report when there are multiple designs.

file Specifies the name of the file to which to write the report.

- subdesign *subdesign*

Specifies the subdesign for which a hierarchical report is required.

Example

- The following example reports the hierarchy of design m1:

```
=====
Hierarchy Report Format :

level instance ( module ) <status>

status :      preserve_<value> -- indicating preserve hierachy or
inherited_preserve value
            :      blackbox -- indicating unresolved instance
=====
```

0 m1
 1 m2 (m2)
 2 m3 (m3)
 3 m3_0 (m3_0)
 4 m3_0_0 (m3_0_0)
 3 m4 (m4)
 4 m5 (m5)
 5 m5_bbox (m5_bbox) blackbox
 4 m6 (m6)
 2 m2myclk (m2myclk)

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

Affected by this attribute: [hdl_track_filename_row_col](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report instance

```
report instance [-timing] [-power] [-detail]
    instance... [> file]
```

Generates a report on the instances of the current design. By default, the report gives timing related information on the instances. Get power related information using the `-power` option.

Options and Arguments

<i>file</i>	Specifies the name of the file to which to write the report.
<code>-detail</code>	Reports information such as pin direction, propagated constants, propagated pins, and disabled timing arcs. Disabled timing arc info is only available for mapped designs.
<i>instance</i>	Specifies the leaf instance for which to generate the report.
<code>-power</code>	Reports instance leakage, internal power, net power and the computed probability, toggle rate, and net power on the nets of the instance, and the activity and power for each of the arcs. In case the switching activities are user-asserted, the values are appended with an asterisk (*). Note: The <code>lp_power_unit</code> attribute does not affect the units of the Arc Energy column.
<code>-timing</code>	Reports timing information, such as slew-in, load, slew-out, delay, and slack.

Example

- The following example reports timing information for the n0000D3666 instance:

```
rc:/> report instance -timing n0000D3666

...
Module:          dpldalg
...
=====
Instance Timing Info
-----
Instance n0000D3666 of libcell nor2

Arc  Arc   Slew      Slew
From To    In    Load   Out   Delay   Slack
-----
A r  Y f   46.2  20.7  57.2  136.0 -1022.4
A f  Y r   46.2  20.7  57.2  136.0 -1022.4
B r  Y f   16.5  20.7  57.2  134.0 -900.2
B f  Y r   16.5  20.7  57.2  134.0 -900.2
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example shows the timing and power information for instance `o_m2_clk2_1_reg_0`.

```
=====
...
Module: m1
Technology library: slow 1.0
Operating conditions: slow (balanced_tree)
Wireload mode: enclosed
=====
Instance m2/o_m2_clk2_1_reg_0 of libcell EDFFX1

Arc Arc Slew      Slew
From To In Load Out Delay Slack
-----
CK   Q  0.0  10.0  173.9  387.0  inf
CK   QN 0.0    0.0    64.3      inf

Instance Power Info
-----
Instance m2/o_m2_clk2_1_reg_0 of Libcell EDFFX1

Leakage Internal Net
Power(nW) Power(nW) Power(nW)
-----
32.822   20652.925   182.250

Computed          Computed          Net
Pin     Net       Probability      Toggle Rate(/ns) Power(nW)
-----
Q      o_m2_clk2_1[0]        0.6        0.0312    182.250
CK     n_0            0.5        1.8724    4913.916
D      in_2[21]         0.5        0.0208    138.510
E      en_2[7]          0.5        0.0208    473.850

Arc Arc          Arc          Arc
From To When Activity(/ns) Energy(fJ)
-----
CK   CK  !D & !E      0.4681    9.396
CK   CK  D & !E      0.4681    9.563
CK   CK  !D & E       0.4681   11.860
CK   CK  D & E       0.4681   11.973
D    D    D           0.0208    7.726
CK   CK  D & E       0.4681   11.973
CK   Q    Q           0.0312    7.902
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example shows a detailed instance report.

```
rc:/> report instance CG1 -detail
=====
...
Module:                                test
Technology library:      mylibraries
Operating conditions:    WCCOM (balanced_tree)
Wireload mode:                  segmented
Area mode:                      timing library
=====

Instance detail information
-----
Instance CG1 of libcell /libraries/mylibraries/libcells/mycell

Pin  Direction  Constant   Clock
-----
E    in
CP   in
TE   in          0
Q    out         clk(+)

Disabled Arcs by constant propagation
-----
/libraries/mylibraries/libcells/mycell/inarcs/CP_TE_Ha0
/libraries/mylibraries/libcells/mycell/TE/inarcs7CP_TE_Sa0

Disabled Arcs by instance based disable_timing
-----

Disabled Arcs by libcell based disable_timing
-----
```

Related Information

[Generating Cell Instance Reports in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

[Reporting Power on Leaf Instances in Low Power in Encounter RTL Compiler](#)

report isolation

```
report isolation
  { [instance_list]
  | [-detail] [-hierarchical]
  [-from_power_domain power_domain...]
  [-to_power_domain power_domain...] }
  [-width integer] [> file]
```

Reports isolation cell information for the design. The return value of the report corresponds to the number of leaf isolation cell instances found. The information returned depends on your current *position* in the design hierarchy.

Options and Arguments

-detail	Requests a detailed isolation cell report. A detailed report shows the names of the hierarchical isolation cell instances, the power domains they are inserted between, the power domain they are stored with, the type of isolation cell, and the number of leaf isolation cell instances they contain. The following types of cells can be distinguished:
	<ul style="list-style-type: none">■ L—Level shifter library cell with isolation logic■ I—Isolation library cell■ D—Discrete isolation cell (can contain an AND gate, or OR gate, or latch, and possibly an inverter)
file	Specifies the name of the file to which to write the report.
-from_power_domain power_domain	Reports all isolation cells whose drivers are output pins of instances in the specified power domains.
-hierarchical	Traverses the hierarchy starting from the current module and reports all the isolation cell instances found in the current module and its children modules.
instance_list	Reports detailed information for the specified hierarchical isolation cell instances. The detailed report lists <ul style="list-style-type: none">■ For the hierarchical instance: the from and to domain, the location, the type of the isolation cells, the enable driver (if it can be determined), and for discrete types, the function.■ For each leaf cell: the driver and the load.

Command Reference for Encounter RTL Compiler

Analysis and Report

Note: The report by instance also shows the type of the isolation cell. In addition to the three types that are distinguished with the detailed report, the report by instance can also distinguish two other types:

- Generic—Refers to a user-specified (isolation) module that contains (an) unmapped cell(s).
- Complex—Refers to a user-specified (isolation) module that contains complex gates, or a hierarchy that contains mixed cell types, such as a pure isolation cell and an enabled level shifter.

These two types cause a warning in the detailed isolation report.

`-to_power_domain power_domain`

Reports all isolation cells whose output pins are driving instances in the specified power domains.

`-width integer`

Specifies the column width to use for the isolation instance names in the report.

By default, the tool automatically resizes the column width for instance names up to 40 characters. If the instance name is longer than 40 characters it will be abbreviated unless you specify the width to be larger.

Examples

- The following command shows the basic report for the top-level design. The design has four power domains, four hierarchical isolation cell instances and four leaf isolation cell instances were inserted.

```
rc:/> report isolation
=====
Generated by:          Encounter(r) RTL Compiler version
Generated on:          date
Module:                top
...
=====
Category
=====
Unique power domains      4
-----
Isolation Cell hierarchical instances 4
Isolation Cell instances      4
=====
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the detailed report for the design and reports all isolation instances found in the hierarchy starting from the top level.

```
rc:/> report isolation -detail -hierarchical

...
Isolation Cell           From      To       Location    Type   Number of
instance     domain    domain
=====
mux_10_14RC_ISO_HIER_INST_22  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_23  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_24  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_26  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_27  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_28  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_29  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_30  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_31  p1       p2       p2          D      1
mux_10_14RC_ISO_HIER_INST_32  p1       p2       p2          D      1
RC_ISO_HIER_INST_19          p3       p4       p1          D      1
RC_ISO_HIER_INST_20          p3       p4       p1          D      1
RC_ISO_HIER_INST_25          p3       p4       p1          D      1
RC_ISO_HIER_INST_21          p3       p2       p3          D      1
                                         p4

-----
Summary
=====
Category
=====
Unique power domains          4
-----
Isolation Cell hierarchical instances 14
Isolation Cell instances        14
=====
```

14

- The following command shows the report for instance RC_ISO_HIER_INST_19.

```
rc:/> report isolation RC_ISO_HIER_INST_19

...
Name:          RC_ISO_HIER_INST_19
From domain(s): p3
To domain(s):  p4
Location:      p1
Type:          Discrete
Function:      Enable: active_high; Output: low
Enable:         s
Details:

  Isolation          Driver          Load
  cell               instance/pin(s)  instance/pin(s)
  instance
=====
  RC_ISO_AND        myInsti/y_reg[2]/Q  y[2]
  RC_ISO_NOT

-----
```

1

Related Information

[Execute CPF File in Low Power in Encounter RTL Compiler](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report level_shifter

```
report level_shifter
  { [instance_list]
  | [-detail] [-hierarchical]
  [-from_power_domain power_domain_list]
  [-to_power_domain power_domain_list] }
  [-verbose] [> file]
```

Reports level-shifter information for the design. The return value of the report corresponds to the number of leaf level-shifter instances found.

Options and Arguments

-detail	Requests a detailed level shifter report.
<i>file</i>	Specifies the name of the file to which to write the report.
-from_power_domain <i>power_domain_list</i>	Reports all level shifters whose drivers are output pins of instances in the specified power domain.
-hierarchical	Reports level shifters hierarchically. If this option is omitted, reports all level shifters at the current level of the hierarchy.
<i>instance_list</i>	Reports detailed information for the specified hierarchical level-shifter instances at the current level.
-to_power_domain <i>power_domain_list</i>	Reports all level shifters whose output pins are driving instances in the specified power domain.
-verbose	Reports complete information if it has been not shown with -detail option.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following command shows the report for the top-level design. The design has three power domains, one hierarchical level-shifter instance stored in the power domain for the top design, and 2 leaf level-shifter instances.

```
rc:/> report level_shifter
=====
Generated by:          Encounter(R) RTL Compiler version
Generated on:          date
Module:                soc
Library domain:        umc_08v
Domain index:          0
Technology libraries:  scmetro_umc1130e_sp_tt_0p8v_25c 1.0
                      scmetro_umc1130e_ll_tt_0p8v_25c 1.0
                      scmetropmk_umc13sp_tt_0p8v_25c 1.0
                      scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
Operating conditions: nominal_(balanced_tree)
Library domain:        umc_120v-
Domain index:          1
Technology libraries:  scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
                      scmetro_umc1130e_ll_tt_1p2v_25c 1.0
                      scmetro_umc1130e_sp_tt_1p2v_25c 1.0
Operating conditions: nominal_(balanced_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====
```

Summary

```
Unique Power Domains : 3
=====
```

Domain Interactions

From Power Domain	To Power Domain	Level Shifter Hierarchies	Level Shifter Instances
PD3 (0.8v-1.2v)	PD1 (0.8v)	1	2
	Total:	1	2
		2	

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following report requests a hierarchical report. This report indicates that the design uses a total of three power domains, has two hierarchical level-shifter instances and 11 leaf level-shifter instances inserted.

```
rc:/> report level_shifter -hier
=====
...
Summary
=====

Unique Power Domains : 3
=====
Domain Interactions
=====
From Power Domain      To Power Domain      Level Shifter      Level Shifter
=====          Hierarchies           Instances
=====
PD1 (0.8v)            PD2 (0.8v-1.2v)    1                4
PD1 (0.8v)            PD3 (0.8v-1.2v)    1                4
PD2 (0.8v-1.2v)       PD3 (0.8v-1.2v)    1                1
PD3 (0.8v-1.2v)       PD1 (0.8v)         1                2
=====
Total:                  4                    11
=====
11
```

- The following command shows the detailed report of the level shifters between power domain PD2 and PD3. The report shows the name of the hierarchical level-shifter instance and the hierarchical instance (module) it was inserted in.

```
rc:/> report level_shifter -from_power PD2 -to_power PD3 -detail -hier
...
=====
From          To          Module          Level           Instances   Location
Power        Power        Module          Shifter
Domain       Domain       Module          Hierarchy
(Range)      (Range)
=====
PD2 (0.8v-1.2v)  PD3 (0.8v-1.2v)  socsync_det  CPF_LS_HIER_INST_27  1          to
=====
```

```
Summary
=====

Unique Power Domains : 3
=====
Domain Interactions
=====
From Power Domain      To Power Domain      Level Shifter      Level Shifter
=====          Hierarchies           Instances
=====
PD2 (0.8v-1.2v)       PD3 (0.8v-1.2v)    1                1
=====
Total:                  1                    1
=====
1
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the report for level-shifter instance CPF_LS_HIER_INST_21. The detailed report lists for each leaf level shifter instance all pins that it is connecting.

```
rc:/> report level_shifter [find / -inst CPF_LS_HIER_INST_21]
...
Name:      cg_pream/CPF_LS_HIER_INST_21
From Power Domain:    PDI (0.8v-0.8v)
To Power Domain:     PD2 (0.8v-1.2v)
Location:          to
Instances:         4
Details:
Level
shifter
instance           From
instance/pin           To
instance/pin           Is
dedicated
=====
g1                  /soc/pd      cg_pream/count_reg[1]/RETN   true
                           cg_pream/count_reg[5]/RETN   true
                           cg_pream/count_reg[9]/RETN   true
                           cg_pream/count_reg[2]/RETN   true
                           cg_pream/count_reg[6]/RETN   true
                           cg_pream/count_reg[3]/RETN   true
                           cg_pream/count_reg[7]/RETN   true
                           cg_pream/prem_ok_reg/RETN  true
                           cg_pream/count_reg[0]/RETN   true
                           cg_pream/count_reg[4]/RETN   true
                           cg_pream/count_reg[8]/RETN   true
                           cg_pream/count_reg[9]/CK    true
                           cg_pream/count_reg[5]/CK    true
                           cg_pream/count_reg[1]/CK    true
                           cg_pream/count_reg[6]/CK    true
                           cg_pream/count_reg[2]/CK    true
                           cg_pream/count_reg[7]/CK    true
                           cg_pream/count_reg[3]/CK    true
                           cg_pream/count_reg[8]/CK    true
                           cg_pream/count_reg[4]/CK    true
                           cg_pream/count_reg[0]/CK    true
                           cg_pream/prem_ok_reg/CK   true
g442                /soc/ck1      cg_pream/g418/B    true
                           cg_pream/g415/A0   true
                           cg_pream/g428/A    true
g443                /soc/rec_en
g444                /soc/rst
```

4

Related Information

[Execute CPF File in Low Power in Encounter RTL Compiler](#)

Affected by these commands: [commit_cpf](#) on page 914

[read_cpf](#) on page 922

Command Reference for Encounter RTL Compiler

Analysis and Report

report memory

report memory [> *file*]

Reports the memory resource used by the compiler in the computing platform.

Options and Arguments

file Specifies the name of the file to which to write the report.

Command Reference for Encounter RTL Compiler

Analysis and Report

report memory_cells

```
report memory_cells [library] [> file]
```

Reports the memory cells in the library.

Options and Arguments

file Specifies the name of the file to which to write the report.

library Specifies the name of the library for which to report the memory cells.

If no library is specified, the report applies to all libraries that are loaded.

Example

The following command lists the memory cells in the RF32X32.lib library:

```

# Generated by: Cadence Encounter(R) RTL Compiler 10.1.100
# Generated on: Feb 23 2010 09:34:16
=====

      Library : RF32X32.lib
      Memory Cell: RF32X32
      Memory Type: ram
      Memory address width: 5
      Memory word width: 32

-----
```

PIN/BUS	DIRECTION	IS_BUS	BUS_TYPE	CAPACITANCE	RELATED_PIN	TIMING_TYPE	TIMING_SENSE
Q	output	31 to 0	YES	RF32X32_DATA		CLK	rising_edge
D	input	31 to 0	YES	RF32X32_DATA	0.002	CLK CLK	hold_rising setup_rising
CEN	input	0 to 0	NO		0.002	CLK CLK	hold_rising setup_rising
WEN	input	0 to 0	NO		0.010	CLK CLK	hold_rising setup_rising
A	input	4 to 0	YES	RF32X32_ADDRESS	0.008	CLK CLK	hold_rising setup_rising

CLOCK	DIRECTION	CAPACITANCE	MAX_TRANSITION	MIN_PULSE_WIDTH_HIGH	MIN_PULSE_WIDTH_LOW	MIN_PERIOD
CLK	input	0.038	0.000	0.055	0.130	0.867

Command Reference for Encounter RTL Compiler

Analysis and Report

report messages

```
report messages [-all] [-include_suppressed] [-error]
                 [-warning] [-info] [> file]
```

Summarizes the information, warning and error messages that have been issued by RTL Compiler in the current run since the last report. The report contains the number of times the message has been issued, the severity of the message, the identification number, and the message text.

The `-all` option does not report those messages that were suppressed through the `suppress_messages` command. Specify the `-include_suppressed` option to report such messages.

By adding report messages to your Tcl prompt, RTL Compiler can summarize all messages issued during the last command right before prompting you for more input. This is useful after long commands (such as `elaborate`) that can generate many messages.

Options and Arguments

<code>-all</code>	Reports all messages since you started this RTL Compiler run.
<code>-error</code>	Reports the error messages.
<code>file</code>	Specifies the name of the file to which to write the report.
<code>-include_suppressed</code>	Reports those messages that were suppressed through the <code>suppress_messages</code> command.
<code>-info</code>	Reports the information messages.
<code>-warning</code>	Reports the warning messages.

Examples

Note: The following examples all apply to the same RTL Compiler session.

- The following example is the first request to report messages in a session.

```
rc:/> report messages
=====
Message Summary
=====

  Num  Sev      Id          Message Text
  ---  --  -----
  1  Info  ELAB-VLOG-9  Variable has no fanout.
                           This variable is not driving anything and will be
                           simplified
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
3 Info LBR-30      Promoting a setup arc to recovery.  
                    Setup arcs to asynchronous input pins are not  
                    supported  
3 Info LBR-31      Promoting a hold arc to removal.  
                    Hold arcs to asynchronous input pins are not  
                    supported  
1 Info LBR-54      Library has missing unit.  
                    Current library has missing unit.
```

- The following example executes the report messages command immediately after the previous report messages command and consequently does not return any new messages.

```
rc:/> report messages
```

- The following example is the third report messages command in a row, but because the -all option is specified, the same output as with the first command is given:

```
rc:/> report messages -all
```

```
=====  
Message Summary  
=====
```

Num	Sev	Id	Message Text
1	Info	ELAB-VLOG-9	Variable has no fanout. This variable is not driving anything and will be simplified
3	Info	LBR-30	Promoting a setup arc to recovery. Setup arcs to asynchronous input pins are not supported
3	Info	LBR-31	Promoting a hold arc to removal. Hold arcs to asynchronous input pins are not supported
1	Info	LBR-54	Library has missing unit. Current library has missing unit.

- The following example requests to print all error messages since this run was started, but no error messages were found:

```
rc:/> report messages -all -error
```

Related Information

[Summarizing Messages in Using Encounter RTL Compiler](#)

report multibit_inferencing

```
report multibit_inferencing
  [-comb] [-seq] [-power]
  [[-instance_hier instance] [design]] | -lib] > file
```

Reports detailed information on the combinational and sequential multibit cells that are used in the design or available in the libraries. Use the options to filter the information. When no options are specified, the report returns the information for all multibit cells used in the design.

Options and Arguments

<code>-comb</code>	Limits the report to combinational multibit cells. For each multibit libcell the following is reported: bitwidth, the number of instances in the design mapped to this libcell, total area of these instances, the multibit conversion ratio (number of single bit instances merged to this multibit cell to the total number of single bit instances), and the library.
<code>design</code>	Specifies the design for which to create the report. If no design is specified, the report is given for the current design. This option is required when multiple designs are loaded. Note: You cannot specify this option together with the <code>-lib</code> option.
<code>file</code>	Redirects the report to the specified file.
<code>-instance_hier <i>instance</i></code>	Prints the report for the specified hierarchical instance.
<code>-lib</code>	Reports the multibit cells available in the libraries. For each libcell, the following is reported: value of the avoid attribute, bitwidth, cell area, cell leakage power, library. Note: This option can be used after the libraries are loaded and before the design is loaded. Note: You cannot specify this option together with the <code>-instance_hier</code> option or when you specify the design.
<code>-power</code>	Reports leakage power information for the multibit cells in the design.

Command Reference for Encounter RTL Compiler

Analysis and Report

-seq

Limits the report to sequential multibit cells.

For each multibit libcell the following is reported: bitwidth, the number of instances in the design mapped to this libcell, total area of these instances, the multibit conversion ratio (number of single bit instances merged to this multibit cell to the total number of single bit instances), and the library.

Examples

- The following command reports the multibit cells in the libraries.

```
rc:/> report multibit_inferencing -lib
=====
Generated by:      Encounter(R) RTL Compiler version
Generated on:      date
Technology libraries: UX8L_widel_1.1V_MAX_primitive_lvt 1.005489
                     UX8L_widel_1.1V_MAX_multibit_lvt 0.000001
Operating conditions: WORST_TREE (worst_case_tree)
Wireload mode:       enclosed
Area mode:          timing library
=====

Library : Combinational Multibit Libcells info
=====

Comb_Mbit libcell  Avoid  Bitwidth  Leakage Power  Area           Library
-----+
LDL_ND2_OP5_2B    false     2        1.23      1.10  UX8L_widel_1.1V_MAX_multibit_lvt

Library : Sequential Multibit Libcells info
=====

Seq_Mbit libcell  Avoid  Bitwidth  Leakage Power  Area           Library
-----+
LDL_FDPQ_1_2B     false     2        14.32     7.53  UX8L_widel_1.1V_MAX_multibit_lvt
LDL_FDPQ_1_4B     false     4        14.32    15.05  UX8L_widel_1.1V_MAX_multibit_lvt
LDL_FDPQ_1_8B     false     8        14.32    30.11  UX8L_widel_1.1V_MAX_multibit_lvt
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports the combinational multibit cells used in the design.

```
rc:/> report multibit_inferencing -comb
=====
Generated by:          Encounter(R) RTL Compiler version
Generated on:          date
Module:                top
Technology libraries: UX8L_widel_1.1V_MAX_primitive_lvt 1.005489
                      UX8L_widel_1.1V_MAX_multibit_lvt 0.000001
Operating conditions: WORST_TREE (worst_case_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====

Combinational Multibit cells usage statistics
=====
Total Combinational instances merged: 2
Total Combinational instances not merged: 0

Comb_Mbit libcell Bitwidth Count Total Area Multibit Conversion % Library
-----
LDL_ND2_OP5_2B           2      1     1.10    100.00   UX8L_widel_1.1V_MAX_multibit_lvt
-----
Total                   1      1.10    100.00
-----
```

- The following command reports the combinational multibit cells used in the design with their leakage power information.

```
rc:/> report multibit_inferencing -comb -power
...
=====
Combinational Multibit cells usage statistics
=====
Total Combinational instances merged: 2
Total Combinational instances not merged: 0

Comb Mbit libcell Bitwidth Count Total Area Total Power(nW) Multibit Conversion % Library
-----
LDL_ND2_OP5_2B           2      1     1.10     1.23    100.00   UX8L_widel_1.1V_MAX_multibit_lvt
-----
Total                     1      1.10     1.23    100.00
-----
```

Related Information

[Mapping to Multibit Cells](#) in Encounter RTL Compiler Synthesis Flows

report net_cap_calculation

```
report net_cap_calculation net [> file]
```

Reports the capacitance values for the different pins or ports that are connected to the specified net.

- For a pin, the capacitance value is the capacitance of the libcell pin (obtained from the .lib).
- For a port, the capacitance value is any capacitance annotated on the port (sum of the `external_pin_cap` and `external_wire_cap` attributes).
- The net capacitance is computed from the wire-load model available in the library. This is based on the `wireload_mode` attribute (top, segmented, or enclosed).
- The final capacitance is the sum of the net capacitance and the pin and port capacitance values to which the net is connected.

The columns "Terms from .lib" and "Cap from .lib" in the report will be populated if the wireload model in the .lib appears as a table. Otherwise, it will be empty.

This command is not supported on those nets that have the `physical_cap` attribute set on them. Also, when you are in PLE mode, only the final capacitance is shown (no computation).

Options and Arguments

file

Redirects the report to the specified file.

net

Specifies the net name for which the report should be generated.

Related Information

Related command:

[report net_res_calculation](#) on page 451

Command Reference for Encounter RTL Compiler

Analysis and Report

report net_delay_calculation

```
report net_delay_calculation [-driver_pin {port|pin}...]
    [-load_pin {port|pin}...] [> file]
```

Reports the net delay, in picoseconds, between the specified driver and load pins. Both the driver and load pins should be on the same net. The delay computed would depend upon the tree type used (`tree_type` attribute): best case, worst case, or balanced tree.

Note: This command assumes that the `interconnect_mode` attribute is set to `wireload`.

Options and Arguments

-driver_pin {port|pin}

Specifies the starting port or pin on which to obtain the net delay.

-load_pin {port|pin}

Specifies the ending port or pin on which to obtain the net delay.

file

Redirects the report to the specified file.

Example

- The following command provides a report based on the `in1[0]` driver pin:

```
rc:/designs/areid/ports_in> report_net_delay_calculation -driver_pin in1[0]
=====
...
Operating conditions:    slow (balanced_tree)
Wireload mode:           segmented
=====
Formula: (Wres/f) * (Pcap + Wcap/f)

From      To          Wire res   Wire cap Fanout Pin cap of  Total pin cap of  Net
pin       pin        of net     of net of net of net to pin      all to pins  delay
-----
in1[0]    inst1/g43/A  0.000     7.2      1       5.3            5.3        0.0
```

Related Information

Affected by this attribute: [tree_type](#)

report net_res_calculation

```
report net_res_calculation net [> file]
```

Reports the total wire resistance of the specified net. The total wire resistance is the sum of the individual net segment resistance (obtained from the wire-load model) and the external wire resistance (annotated on any port and obtained from the `external_wire_res` attribute) that is connected to the net.

- For a port, the resistance value is any resistance annotated on the port (the `external_wire_cap` attributes).
- The net resistance is computed from the wire-load model available in the library. This is based on the `wireload_mode` attribute (top, segmented, or enclosed).
- The final resistance is the sum of the net resistance and the pin and port resistance values to which the net is connected.

The columns "Terms from .lib" and "Cap from .lib" in the report will be populated if the wire-load model in the .lib appears as a table. Otherwise, it will be empty.

This command is not supported on those nets that have the `physical_res` attribute set on them. Also, when you are in PLE mode, only the final resistance is shown (no computation).

Options and Arguments

file

Redirects the report to the specified file.

net

Specifies the net name for which the report should be generated.

Related Information

Related command:

[report net_cap_calculation](#) on page 449

report nets

```
report nets [-hierarchical] [-pin pin...]
            [-minfanout integer] [-maxfanout integer]
            [net | instance]... [-sort string]
            [-cap_worst integer] [> file]
```

Generates a report on the nets of the current design. The report gives information for the top-level nets in the design. You can specify pin names, nets, instances, maximum and minimum fanout threshold values, nets, and instances. Control the data printed out using the -minfanout and -maxfanout options for nets that have fanout between these values. Nets that are followed by the "@" symbol in parenthesis indicate SPEF annotation.

Options and Arguments

-cap_worst <i>integer</i>	Specifies the number of worst capacitance nets that are to be reported.
<i>file</i>	Specifies the name of the file to which to write the report.
-hierarchical	Reports all the nets in the design hierarchy.
-maxfanout	Specifies an <i>integer</i> value and reports nets whose fanouts are below the given threshold value.
-minfanout	Specifies an <i>integer</i> value and reports nets whose fanouts are above the given threshold value.
<i>net</i> <i>instance</i>	Reports information on the specified nets or nets belonging to the instance.
-pin <i>pin</i>	Specifies a list of pin names and reports the nets connected to the pins.
-sort	Specifies the field name on which to sort. Valid field names are load, resistance, or capacitance.

Examples

- The following example shows that the address nets are SPEF annotated while the accum nets are not:

```
=====
...
=====
address[0] (@)      1      1      1.0      0.000
address[1] (@)      1      1      0.8      0.000
accum[0]           1      1      2.1      0.000
accum[1]           1      1      8.5      0.000
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports the five worst capacitance nets in the top level:

```
rc:\> report nets -cap_worst 5
=====
Generated by:          RTL Compiler (RC) Version
Generated on:          Date
Module:                m1
Technology library:   slow 1.5
Operating conditions: slow (balanced_tree)
Wireload mode:        segmented
=====

          Wire      Wire      Wireload
Net    Loads   Drivers Cap(fF) Res(k-ohm) Model
-----
ai      2       3       14.5    0.000
n_135  2       2       10.4    0.000
n_0     2       1       6.7     0.000
ao[5]   1       2       6.7     0.000
bi      1       1       3.6     0.000
=====
```

- The following example sorts the nets in the top level by the number of loads:

```
rc:\> report nets -sort load
=====
...
=====
          Wire      Wire      Wireload
Net    Loads   Drivers Cap(fF) Res(k-ohm) Model
-----
n_1    8       1
c\k    3       1       0.0     0.000
n_135 2       2       10.4    0.000
n_0    2       1       6.7     0.000
ai     2       3       14.5    0.000
.....
=====
```

- The following example provides specific information on the n_0 ai net:

```
rc:\> report net n_0 ai
=====
...
=====
      Total    Slew    Slew
Net Cap(fF)  Rise Fall Driver(s)  Load(s)
-----
n_0    20.3   0.0   0.0  g24/Y    g23/A
                           g22/A
                           bx_reg3/CK
ai     12.0   0.0   0.0  ai       g25/A
                           bx_reg2/D
=====
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports all the nets in the top level of the current design whose fanout is less than 2.

```
rc:> report net -maxfanout 2
=====
...
=====
```

Net	Loads	Drivers	Wire Cap(fF)	Wire Res(k-ohm)	Wireload Model
in1a[0]	1	1	0.4	0.000	AL_SMALL
in1a[1]	1	1	0.4	0.000	AL_SMALL
in1b[0]	1	1	0.4	0.000	AL_SMALL
in1b[1]	1	1	0.4	0.000	AL_SMALL
out2a[0]	1	1	0.4	0.000	AL_SMALL
out2a[1]	1	1	0.4	0.000	AL_SMALL
out2b[0]	1	1	0.4	0.000	AL_SMALL
out2b[1]	1	1	0.4	0.000	AL_SMALL
out3a[0]	1	1	0.0	0.000	AL_SMALL
out3a[1]	1	1	0.0	0.000	AL_SMALL
out3b[0]	1	1	0.0	0.000	AL_SMALL
out3b[1]	1	1	0.0	0.000	AL_SMALL
out4a[0]	1	1	0.0	0.000	AL_SMALL
out4a[1]	1	1	0.0	0.000	AL_SMALL
out4b[0]	1	1	0.0	0.000	AL_SMALL
out4b[1]	1	1	0.0	0.000	AL_SMALL

```
=====
```

You can specify an instance name to the above example and get information on the nets associated with the instance(s) whose fanout is less than 2.

- The following example reports the nets associated with the g22/A bx_reg2/D pin:

```
rc:/> report net -pin "g22/A bx_reg2/D"
=====
...
=====
```

Total Net Cap(fF)	Slew Rise	Slew Fall	Driver(s)	Load(s)
n_0	20.3	0.0	0.0	g24/Y
				g23/A
				g22/A
				bx_reg3/CK
ai	12.0	0.0	0.0	ai
				g25/A
				bx_reg2/D

```
=====
```

Related Information

[Generating a Net Report in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

report opcg_equivalents

```
report opcg_equivalents  
    [-pinVal] [design] [> file]
```

Reports the OPCG-equivalency mappings specified using the `set_opcg_equivalent` command(s).

Options and Arguments

<i>design</i>	Specifies the design for which you want to report the OPCG-equivalency mappings.
<i>file</i>	Specifies the name of the file to which to write the report.
<code>-pinVal</code>	Prints the pin constant values.

Example

The following report shows two OPCG-equivalency mappings. For each mapping, the report lists the scan cell, the equivalent OPCG cell, the edge-mode pin, the loopback pin, and the pin mappings between the two cells.

```
rc:/> report opcg_equivalents  
OPCG equivalent table contains 2 entries {  
{  
    Scan cell: /libraries/myspecial/libcells/SDFFQN_X1M  
    Opcg cell: /libraries/myspecial/libcells/S2DFFQQN_X1M  
    Edge_mode: TEL  
    Loop_back: TI  
    Pin map: {SE SE} {SI SI} {D D} {CK CK} {QN QN}  
}  
  
{  
    Scan cell: /libraries/myspecial/libcells/SDFFQ_X1M  
    Opcg cell: /libraries/myspecial/libcells/S2DFFQQN_X1M  
    Edge_mode: TEL  
    Loop_back: TI  
    Pin map: {D D} {CK CK} {Q Q} {SE SE} {SI SI}  
}  
}
```

Related Information

Affected by these commands: [reset_opcg_equivalent](#) on page 795
[set_opcg_equivalent](#) on page 799

Related commands: [replace_opcg_scan](#) on page 786

Command Reference for Encounter RTL Compiler

Analysis and Report

report operand_isolation

```
report operand_isolation  
[-oi_instance instance...] [> file]
```

Reports operand-isolation information for the design. The information depends on whether the operand-isolation logic has been committed or not.

Note: This command only reports operand-isolation logic inserted by the RC-LP engine, provided that you do not remove the subdesigns corresponding to the operand-isolation logic. It does not report operand-isolation logic inserted by third-party tools.

Options and Arguments

-oi_instance instance

Reports detailed information for the specified operand-isolation instances. Information includes the name of the isolated instance, the list of control inputs and their associated switching activities, the list of data inputs that have been isolated, the list of output pins of the operand-isolation instance.

Note: If the isolated instance name is (flattened), the datapath instance was flattened during optimization.

file

Specifies the name of the file to which to write the report.

Examples

- The following command gives a summary after operand-isolation logic has been inserted.

```
rc:/> report operand_isolation  
=====  
Generated by:          RTL Compiler-D (RC) version  
....  
=====  
  
Operand Isolation Instances  
-----  
Module  Operand Isolation Instance  Width  Isolated Instance  Control Inputs  
-----  
test    RC_OI_HIER_INST           8      add_13_21          en1, en2  
        RC_OI_HIER_INST6          8      add_13_21          en1, en2  
-----  
Total    2  
=====
```

Command Reference for Encounter RTL Compiler

Analysis and Report

where

- ❑ *Width* is the width of the data_bus input of the operand-isolation instance
- ❑ *Isolated instance* is the datapath instance whose input is isolated.
- ❑ *Control inputs* lists the control signals that are inputs to the operand-isolation instance

- The following command generates detailed operand-isolation instance information for the specified operand-isolation instance:

```
rc:/> report operand_isolation -oi_instance RC_OI_HIER_INST
=====
Generated by:          RTL Compiler-D (RC) version
...
=====
Operand Isolation Instance : RC_OI_HIER_INST
-----
Module:                test (test)
Isolated Instance:    add_13_21 (test/add_13_21)
Control Inputs:
  RC_OI_CTRL_PORT      =      en1 (/designs/test/ports_in/en1)
                           TCF = (0.50000, 0.0200007ns)
  RC_OI_CTRL_PORT_1    =      en2 (/designs/test/ports_in/en2)
                           TCF = (0.50000, 0.0200007ns)
Data Inputs:
  RC_OI_DATA_PORT      =      in1[7]  (/designs/test/ports_in/in1[7])
                           in1[6]  (/designs/test/ports_in/in1[6])
                           in1[5]  (/designs/test/ports_in/in1[5])
                           in1[4]  (/designs/test/ports_in/in1[4])
                           in1[3]  (/designs/test/ports_in/in1[3])
                           in1[2]  (/designs/test/ports_in/in1[2])
                           in1[1]  (/designs/test/ports_in/in1[1])
                           in1[0]  (/designs/test/ports_in/in1[0])
Outputs:
  RC_OI_OUT_PORT       =      n_79  (/designs/test/i..13_21/pins_in/A[7])
                           n_78  (/designs/test/i..13_21/pins_in/A[6])
                           n_77  (/designs/test/i..13_21/pins_in/A[5])
                           n_76  (/designs/test/i..13_21/pins_in/A[4])
                           n_75  (/designs/test/i..13_21/pins_in/A[3])
                           n_74  (/designs/test/i..13_21/pins_in/A[2])
                           n_73  (/designs/test/i..13_21/pins_in/A[1])
                           n_72  (/designs/test/i..13_21/pins_in/A[0])
```

Related Information

[Reporting Operand Isolation Information in Low Power in Encounter RTL Compiler](#)

Affected by this command: [synthesize](#) on page 348

Affected by these attributes: [lp_operand_isolation_prefix](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report ple

report ple [*design*]... [> *file*]

Returns the physical layout estimation information for the specified design. The command reports information like aspect ratio, shrink factor, site size, layer names, direction of layers (**Horizontal**, **Vertical** or **Undefined**), layer utilization, capacitance, resistance, area, and the source used to extract the physical information.

Options and Arguments

design Specifies the design name on which to report. If no design is specified, the current design is loaded.

file Specifies the name of the file to which to write the report.

Example

- The following example reports the ple information for the current design:

```
rc:/> report ple
=====
Generated by:          Encounter(R) RTL Compiler 10.1.100
Generated on:         Apr 30 2010  03:29:32 pm
Module:               DTMF_CHIP
Technology libraries: tsmc18 1.0
                      tpz973g 230
                      pllclk 4.3
                      ram_128x16A 1.1
                      ram_256x16A 1.1
                      rom_512x16A 1.1
                      physical_cells
Operating conditions: slow
Interconnect mode:   global
Area mode:           physical library
=====
Aspect ratio       : 1.00
Shrink factor     : 1.00
Scale of res/length : 1.00
Scale of cap/length : 1.00
Net derating factor : 1.00
Site size          : 5.70 um (from lef [tech+cell])
```

Layer Name	Direction	Utilization	Capacitance / Length (pF/micron)	Data source: cap_table_file
M1	H	1.00	0.000274	
M2	V	1.00	0.000242	
M3	H	1.00	0.000242	
M4	V	1.00	0.000242	
M5	H	1.00	0.000242	

Command Reference for Encounter RTL Compiler

Analysis and Report

M6	V	1.00	0.000304	
Layer Name	Direction	Utilization	Resistance / Length (ohm/micron)	Data source: lef_library
Metal1	H	1.00	0.439130	
Metal2	V	1.00	0.360714	
Metal3	H	1.00	0.360714	
Metal4	V	1.00	0.360714	
Metal5	H	1.00	0.360714	
Metal6	V	1.00	0.102273	
Layer Name	Direction	Utilization	Area / Length (micron)	Data source: lef_library
Metal1	H	1.00	0.230000	
Metal2	V	1.00	0.280000	
Metal3	H	1.00	0.280000	
Metal4	V	1.00	0.280000	
Metal5	H	1.00	0.280000	
Metal6	V	1.00	0.440000	

Related Information

“Checking the Physical Layout Estimation Information” in [Simple PLE Flow](#), [Spatial Flow](#), and [RC-P Flow](#) in *Design with RTL Compiler Physical*

Command Reference for Encounter RTL Compiler

Analysis and Report

report port

```
report port [-delay] [-driver] [-load] port... [> file]
```

Generates reports on the ports of the current design. By default, the report gives information on port direction, external delays, exception objects and their types, driver, slew, fanout load, pin capacitance and wire capacitance for the ports. You can also specify the port names on which the report is to be generated and control the data printed using the *-delay*, *-driver* and *-load* options.

Options and Arguments

<i>-delay</i>	Reports external delay information (rise and fall delay and the external delay object)
<i>-driver</i>	Reports the external driver name and the slew (rise and fall) values of the ports.
<i>file</i>	Specifies the name of the file to which to write the report.
<i>-load</i>	Reports the external fanout load and the pin and wire capacitance values of the ports.
<i>port</i>	Specifies the port for which to generate the report.

Example

The following example reports the external delay information on the *ck1*, *e_out[6]*, and *ena* ports:

```
rc:/> report port -delay -driver -load ck1 e_out[6] ena
External Delays & Exceptions
-----
      Port   Dir   Clock   Rise   Fall   Ext Delay       Exception
                Delay   Delay   Object   Object/Type
-----
    ck1     in    CLK1    700.0  700.0  in_del_1           N/A
            CLK2    500.0  500.0  in_del_2
    e_out[6]  out   CLK1    200.0  200.0  ou_del_1  del_1 (path_delay)
            CLK2    300.0  300.0  ou_del_2
                    300.0  300.0  ouTrxtI
    ena     inout  CLK1    700.0  700.0  in_del_1           N/A
            CLK2    500.0  500.0  in_del_2
    ena     inout  CLK1    200.0  200.0  ou_del_1  del_1 (path_delay)
            CLK2    300.0  300.0  ou_del_2
                    300.0  300.0  ouTrxtI
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

[Generating a Port Report in Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report power

```
report power
  { -rtl_cross_reference [-detail]
    [-flat [-nworst number]] [-sort mode]
    [design | instance]...
    [-power_mode power_mode] [-tcf_summary]
  | [-hier | -flat [-nworst number]] [-depth number]
    [-sort mode] [design | instance | net]...
    [-power_mode power_mode] [-tcf_summary]
  | -clock_tree [clock]...
    -width float -height float }
  [-verbose] [> file]
```

Reports the power consumed. The information returned depends on your current *position* in the design hierarchy and on the specified objects. If no objects are specified, the report is given for the design or instance at the current *position* in the design hierarchy.

With the `-rtl_cross_reference` option specified, the power consumed by the instances is cross-referenced to the corresponding line in the RTL files. Set the `hdl_track_filename_row_col` attribute to `true` before elaboration to enable filename, column, and line number tracking.

Note: Nets connected to primary inputs and outputs are only reported at the top-level of an instance-based power report.

Options and Arguments

<i>clock</i>	Specifies the name of a clock for which you want to estimate the clock tree power. If no clock is specified, the power is estimated for all clocks in the design.
<i>-clock_tree</i>	Estimates the power of the clock tree. You can use this option with generic and mapped netlists.
<i>-depth number</i>	Specifies the number of hierarchy levels to descend in the report. If an instance is specified, the depth starts from the position of that instance in the design hierarchy. Use a non-negative integer. Note: This option applies to instance-based power reports, but is not supported with the <i>-rtl_cross_reference</i> option. Default: infinite (all levels of the hierarchy)

Command Reference for Encounter RTL Compiler

Analysis and Report

<i>design</i>	Specifies the design for which you want the power to be reported. Specify the path name to the design. If your current <i>position</i> in the design hierarchy is at the design level and you have only one design loaded, the report is by default given for the design.
<i>-detail</i>	Adds an abbreviated version of the RTL line and a list of the instances that correspond to that RTL line. In this report, the dynamic power is replaced with the internal power and net power (the two components of dynamic power). The detailed report also returns the power information for the primary inputs. Note: This option only applies if you specified the <i>-rtl_cross_reference</i> option.
<i>file</i>	Specifies the name of the file to which to write the report.
<i>-flat</i>	Reports the power of leaf instances (combinational and sequential instances) starting from the <i>current</i> position in the hierarchy. <ul style="list-style-type: none">■ If you perform a gate-level power analysis, the number of hierarchical levels that are expanded depends on the setting of the <i>-depth</i> option.■ If you perform RTL power analysis using the <i>-rtl_cross_reference</i> option, power information for all modules in the current hierarchy is shown. Note: This option only applies to instance-based power reports.
<i>-full_instance_names</i>	Reports the full path names of the instances.
<i>-height float</i>	Specifies the estimated chip height (in microns). Note: This option can only be specified with the <i>-clock_tree</i> option and is optional if a DEF file was read in.
<i>-hier</i>	Reports the power of hierarchical instances. The number of hierarchical levels shown depends on the setting of the <i>-depth</i> option. If you specify neither the <i>-flat</i> or <i>-hier</i> option, the RC-LP engine uses the <i>-hier</i> option by default. Note: This option applies to instance-based power reports, but is not supported with the <i>-rtl_cross_reference</i> option.

Command Reference for Encounter RTL Compiler

Analysis and Report

<i>instance</i>	Specifies the instance for which you want the power to be reported. Specify the path name to the instance.				
<i>net</i>	Specifies the net for which you want the power to be reported. Specify the path name to the net. Note: Net-based power reports are not supported with the <code>-rtl_cross_reference</code> option.				
<code>-nworst number</code>	Prints only the top worst entries of a sorted report. This option applies only to instance-based reports, and can only be used with the <code>-flat</code> option.				
<code>-power_mode power_mode</code>	Only prints the power report for the specified power mode. The specified power mode must correspond to one of the power mode names defined with a <code>create_power_mode</code> command in the CPF file that was read in. If the power mode does not exist, the command will look for a (timing constraint) mode with that name. If you have a multi-mode design and you omit this option, the report will apply to the current state of the design. Note: This option cannot be used with the <code>-clock_tree</code> option.				
<code>-rtl_cross_reference</code>	Cross-references the power consumed to the corresponding line in the RTL files. The report also returns the leakage power, dynamic power, and total power for the top-level design.				
<code>-sort mode</code>	Indicates how to sort the report. The following modes are available for <i>instance</i> -based reports: <table><tr><td><code>dynamic</code></td><td>Sorts by descending total dynamic power, which is the sum of the internal and net power. Note: This option is not supported with the <code>-rtl_cross_reference</code> option.</td></tr><tr><td><code>file</code></td><td>Sorts by RTL file and line number.</td></tr></table>	<code>dynamic</code>	Sorts by descending total dynamic power, which is the sum of the internal and net power. Note: This option is not supported with the <code>-rtl_cross_reference</code> option.	<code>file</code>	Sorts by RTL file and line number.
<code>dynamic</code>	Sorts by descending total dynamic power, which is the sum of the internal and net power. Note: This option is not supported with the <code>-rtl_cross_reference</code> option.				
<code>file</code>	Sorts by RTL file and line number.				

Command Reference for Encounter RTL Compiler

Analysis and Report

Note: This option applies only to RTL power analysis and is the default for RTL power analysis.

internal Sorts by descending internal power.

leakage (default) Sorts by descending leakage power.

net Sorts by descending net power.

The following modes are available for *net*-based reports:

dynamic (default) Sorts by descending total dynamic power.

load Sorts by descending capacitive load on the net.

net Sorts by descending total switching power.

prob Sorts by descending probability of the nets being high.

rate Sorts by descending toggle rates of the nets.

Note: If a report is requested for nets and instances, but the specified sort mode applies only to one category, the other category will be sorted according to its default.

-tcf_summary

Adds a summary to the power report listing the following:

- The number of primary inputs asserted in the design.
- The total number of primary inputs connected in the design.
- The number of sequential outputs asserted in the design.
- The total number of sequential outputs connected in the design.
- The total number of nets in the design.
- *Nets asserted* refer to nets with asserted switching activities (probability and toggle rate).
- *Asserted clock nets* refer to nets whose `lp_probability_type` or `lp_toggle_rate_type` attribute value is set to `clock`.

Command Reference for Encounter RTL Compiler

Analysis and Report

- *Constant nets* refer to nets whose driver is either a constant object 0 or 1.

For net-based reports, an asterisk (*) is appended to each net that has user-asserted switching activities.

`-verbose`

Replaces the dynamic power column with the components of the dynamic power—the internal power and net power.

`-width float`

Specifies the estimated chip width (in microns).

Note: This option can only be specified with the `-clock_tree` option and is optional if a DEF file was read in.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following command requests a basic RTL power analysis of design mult_bit_muxed_add.

```
rc:/> build_rtl_power_models -clean_up_netlist
rc:/> report power -rtl_cross_reference
=====
...
Technology library:    xxx yy
Operating conditions: _nominal_ (balanced_tree)
Wireload mode:         enclosed
=====

Design          Leakage   Dynamic   Total
      Power(nW)  Power(nW)  Power(nW)
-----
mult_bit_muxed_add    71.146  1578.273 1649.419

File           Row   Leakage   Dynamic   Total
      Power(nW)  Power(nW)  Power(nW)
-----
mult_bit_muxed_add.v  8     35.573   665.277  700.850
mult_bit_muxed_add.v  9     5.573    675.858   711.431
```

- The following command shows RTL power analysis for all levels of the hierarchy:

```
rc:/> report power -rtl -flat
=====
...
Technology library:    xxx yy
Operating conditions: _nominal_ (balanced_tree)
Wireload mode:         enclosed
=====

Design          Leakage   Dynamic   Total
      Power(nW)  Power(nW)  Power(nW)
-----
mult_bit_muxed_add    71.146  1578.273 1649.419

File           Row   Leakage   Dynamic   Total
      Power(nW)  Power(nW)  Power(nW)
-----
muxed_add.v      8     28.308   519.317  547.625
muxed_add.v      9     21.419   403.153  424.572
muxed_add.v     12     21.419   418.665  440.084
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command requests a detailed RTL power analysis report.

```
rc:/> report power -rtl -detail -flat
=====
...
Module:          mult_bit_muxed_add
...
=====
Design          Leakage   Internal   Net
              Power(nW) Power(nW) Power(nW)
-----
mult_bit_muxed_add      71.146  1069.140  509.134
-----
Primary Input          Leakage   Internal   Net
              Power(nW) Power(nW) Power(nW)
-----
a[1]                  0.000    0.000    19.428
a[0]                  0.000    0.000    19.428
b[1]                  0.000    0.000    19.428
b[0]                  0.000    0.000    19.428
c[1]                  0.000    0.000    19.428
c[0]                  0.000    0.000    19.428
d[1]                  0.000    0.000    19.428
d[0]                  0.000    0.000    19.428
s                     0.000    0.000    81.713
-----
File       Row     RTL Line Instances  Leakage   Internal   Net
              Power(nW) Power(nW) Power(nW)
-----
muxed_add.v    8     {if (s) begin}      g1      28.308  397.891  121.426
                g1
                g1
                g1
muxed_add.v    9     {y = a + c;}      g1      21.419  330.297  72.856
                g1
muxed_add.v   12     {y = b + d;}      g1      21.419  340.952  77.713
                g1
-----
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command requests a detailed RTL power analysis report for instance add_9_15.

```
rc:/> report power -rtl -detail [find . -inst add_9_15]
=====
...  

Module:          mult_bit_muxed_add  

...  

=====

Design          Leakage   Internal   Net
                Power(nW) Power(nW) Power(nW)
-----
mult_bit_muxed_add      71.146  1069.140  509.134
-----  

Primary Input        Leakage   Internal   Net
                Power(nW) Power(nW) Power(nW)
-----
a[1]              0.000    0.000    19.428
a[0]              0.000    0.000    19.428
b[1]              0.000    0.000    19.428
b[0]              0.000    0.000    19.428
c[1]              0.000    0.000    19.428
c[0]              0.000    0.000    19.428
d[1]              0.000    0.000    19.428
d[0]              0.000    0.000    19.428
s                 0.000    0.000    81.713
-----  

File             Row    RTL Line Instances   Leakage   Internal   Net
                Power(nW) Power(nW) Power(nW)
-----
muxed_add.v     9     y = a + c; add_9_15      21.419  330.297  72.856
                  add_9_15
-----
```

Note: The report shows two add_9_15 instances because RTL Compiler found two instances with that name.

- The following example first descends in the design hierarchy down to instance add_9_15, then requests a detailed RTL power analysis. The output is slightly different from the previous example.

```
rc:/> cd /designs/mult_bit*/instances_hier/ma0/instances_hier/add_9_15
rc:/designs/mult_bit_muxed_add/instances_hier/ma0/instances_hier/add_9_15>
report power -rtl -detail
=====
...  

=====

File             Row    RTL Line Instances   Leakage   Internal   Net
                Power(nW) Power(nW) Power(nW)
-----
muxed_add.v     9     y = a + c; add_9_15      10.709  170.476  38.856
-----
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the clock tree power estimation for clock clk.

```
rc:/> report power -clock_tree iCLK1 -width 5 -height 5
=====
Generated by:          Encounter(r) RTL Compiler version
Generated on:          date
Module:                test
Technology libraries: typical 1.3
Operating conditions: typical (balanced_tree)
Wireload mode:         segmented
Area mode:             timing library
=====

Clock Power Estimation Summary for clock 'iCLK1'
=====

-----
Estimate    Leakage (nW)    Dynamic (nW)    Total (nW)
-----
Max          0.007        147560.871      147560.878
Min          0.007        42160.249       42160.251
Typical     0.007        67941.241       67941.244

Leaf CGIC Cells           4
Leaf Clock Buffers        0
Total Clock Buffers       2

Estimation Parameters
=====

Clock Buffers Used: BUFX12 BUFX16 BUFX2
                     BUFX20 BUFX3 BUFX4
                     BUFX6 BUFX8 CLKBUFX12
                     CLKBUFX16 CLKBUFX2 CLKBUFX20
                     CLKBUFX3 CLKBUFX4 CLKBUFX6
                     CLKBUFX8 DLY1X1 DLY1X4
                     DLY2X1 DLY2X4 DLY3X1
                     DLY3X4 DLY4X1 DLY4X4

Max flops driven by one leaf buffer: 3
Die width: 5.0 um
Die height: 5.0 um
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design. With the -hier option specified, the report lists the design and its hierarchical instances.

```
rc:/designs/mult_bit_muxed_add> report power -hier
=====
...
=====

              Leakage      Dynamic      Total
Instance      Cells Power (nW)  Power (nW)  Power (nW)
-----
mult_bit_muxed_add      6    71.146   1578.273   1649.419
                         ma0      3    35.573   636.281    671.854
                         ma1      3    35.573   638.728    674.301
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design. With the `-verbose` option specified, the report shows in addition the components of the dynamic power.

```
rc:/designs/mult_bit_muxed_add> report power -verbose
=====
...
=====
          Leakage   Internal    Net      Dynamic      Total
          (Int+Net) (Leak+Dyn)
  Instance    Cells Power (nW)  Power (nW)  Power (nW)  Power (nW)
-----
mult_bit_muxed_add    6    71.146  1069.140  509.134  1578.273  1649.419
  ma0            3    35.573   533.055  103.226   636.281   671.854
  ma1            3    35.573   536.085  102.643   638.728   674.301
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is the design. With the `-flat` option specified, the report lists leaf instances starting from the current position in the hierarchy.

```
rc:/designs/mult_bit_muxed_add> report power -flat
=====
...
=====
          Leakage   Dynamic      Total
  Instance Cells Power (nW)  Power (nW)  Power (nW)
-----
  ma0/g38        13.900  255.674  269.574
  ma0/g39        13.900  253.342  267.242
  ma1/g38        13.900  223.699  237.599
  ma1/g39        13.900  253.350  267.250
  ma0/g37        7.774   127.265  135.039
  ma1/g37        7.774   161.679  169.453
```

- The following command reports the power (after synthesis) at the current level in the hierarchy, which is a subdesign. With the `-hier` option specified, the report lists the subdesign and its hierarchical instances. Because in this case there are no hierarchical instances, only the power for the subdesign is listed.

```
rc:/designs/mult_bit_muxed_add/instances_hier/ma0> report power -hier
=====
...
=====
          Leakage   Dynamic      Total
  Instance Cells Power (nW)  Power (nW)  Power (nW)
-----
  ma0            3    35.573   636.281   671.854
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports the power for an instance and sorts the report according to descending net power.

```
rc:/designs/mult_bit_muxed_add> report power -flat instances_hier/ma0 \
-sort net
=====
...
=====

      Leakage    Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
-----
ma0/g39          13.900   255.674   269.574
ma0/g38          13.900   253.342   267.242
ma0/g37          7.774    127.265   135.039
```

- The following command reports the power for an instance and a net.

```
rc:/designs/mult_bit_muxed_add> report power nets/a[1] ma0 -flat
=====
...
=====

      Leakage    Dynamic    Total
Instance Cells Power(nW) Power(nW) Power(nW)
-----
ma0/g39          13.900   255.674   269.574
ma0/g38          13.900   253.342   267.242
ma0/g37          7.774    127.265   135.039

      Net        Net        Toggle
      (asserted *) Power (nW) Prob. Rate (/ns) Cap. (nF)
-----
a[1]            26.827  0.500       0.020     2.300
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following design has five power domains and three power modes. The following commands show the power report for two of the modes after mapping. In this case the report has an additional column Domain (voltage) which shows for each instance to which power domain it belongs and what the voltage is of the domain in the reported mode. Note that when the second report power command is given, the tool adjusts the wireload models for the specified mode before reporting power.

```
rc:/designs/counter> report power -power_mode PMdefault
=====
...
```

```
Module:          counter
Library domain: umc_0p8v
Domain index:   0
Technology libraries: ...
Operating conditions: _nominal_ (balanced_tree)
Library domain: umc_1v
Domain index:   1
...
Library domain: umc_1p2v
Domain index:   2
...
Wireload mode:  enclosed
Area mode:      timing library
Power mode:    PMdefault
=====
```

Instance	Domain (Voltage)	Leakage Cells Power(nW)	Dynamic Power(nW)	Total Power(nW)
counter	PDcore(0.81v)	142 7.076	9192.022	9199.099
monitor_power	PDmon(0.81v)	84 3.720	4050.236	4053.956
adder_counter	PDadd(0.81v)	25 1.215	1573.961	1575.175
bcd_counter	PDbcd(0.81v)	12 1.088	1697.572	1698.659
binary_counter	PDbin(0.81v)	18 1.012	1565.877	1566.888

```
rc:/designs/counter> report power -power_mode PMmid
=====
```

```
...
Power mode:    PMmid
=====
```

```
Applying wireload models.
Info : Changing wireload model of a design/subdesign. [TIM-92]
      : Changing wireload model of design 'counter' from <none> to cmos065.
      : The change of wireload model will likely change the design's timing
slightly.
```

```
...
    Computing net loads.
```

Instance	Library Domain	Leakage Cells Power(nW)	Dynamic Power(nW)	Total Power(nW)
counter	umc_1v	142 11.023	33056.713	33067.735
monitor_power	umc_1v	84 5.805	14774.642	14780.447
adder_counter	umc_1v	25 1.884	4185.202	4187.086
bcd_counter	umc_1v	12 1.697	6138.779	6140.477
binary_counter	umc_1v	18 1.569	6542.145	6543.714

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [RTL Power Analysis](#)
- [Reporting Clock Tree Power](#)
- [Reporting on All Power Components](#)
- [Reporting Leakage Power](#)
- [Reporting Dynamic Power](#)

Affected by these commands: [build_rtl_power_models](#) on page 861
 [synthesize](#) on page 348

Affected by these attributes: [cell_leakage_power](#)
 [disable_power_mode_factorization](#)
 [leakage_power_scale_in_nW](#)
 [lp_asserted_probability](#)
 [lp_asserted_toggle_rate](#)
 [lp_power_unit](#)

Related attributes [lp_clock_tree_buffers](#)
 [lp_clock_tree_leaf_max_fanout](#)
 [lp_computed_probability](#)
 [lp_computed_toggle_rate](#)
 [lp_leakage_power](#)

report power_domain

```
report power_domain  
  [-detail] [-power_mode] [-qor]  
  [> file]
```

Reports power domain related information.

Note: An asterisk (*) identifies the default power domain.

Without any options specified, a summary report is given which shows for each power domain

- The name of the shutoff signal
- The polarity of the shutoff signal
- The operating voltage in the default power mode

Options and Arguments

-detail Prints the summary information and the information you would get by specifying the **-mode**, **-power_nets** and **-qor** options.

file Specifies the name of the file to which the report is to be written.

-power_mode Prints the operating voltage of each power domain in each power mode.

-qor Prints the following information for each power domain:

- The cell area
- The percentage of nets with user-asserted switching activities
- The dynamic power consumption
- The leakage power consumption

Note: The results are given for the current power mode and are affected by the setting of the **lp_power_unit** attribute.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example shows the basic report (with no options specified).

```
rc:/designs/top> report power_domain
Summary
=====
=====
          Shut-off           signal
-----
Name      Name      Active level   Voltage (V)
=====
LD1       -         -             1.2
PD1 (*)  -         -             0.8
PD2       pm_inst/pse_enable[0] active_high  0.8
PD3       pm_inst/pse_enable[1] active_high  0.8
PD4       pm_inst/pse_enable[2] active_high  0.8
-----
5
```

- The following example shows the power mode information. The default power mode is marked with an asterisk.

```
rc:/designs/top> report power_domain -power_mode
Power Modes
=====
=====
          Power Modes
-----
Power Domain PM1     PM2     PM3     PM4
=====
LD1          1.2     1.2     1.2     1.2
PD1 (*)     0.8     0.8     0.8     0.8
PD2          0.8     OFF     OFF     OFF
PD3          0.8     0.8     OFF     OFF
PD4          0.8     0.8     0.8     OFF
-----
5
```

Related Information

[Read CPF File in Low Power in Encounter RTL Compiler](#)

Affected by these commands: [read_cpf](#) on page 922

Command Reference for Encounter RTL Compiler

Analysis and Report

report qor

```
report qor
  [-levels_of_logic] [-nopower]
  [design]... [> file]
```

Reports the critical path slack, total negative slack (TNS), number of gates on the critical path, and number of violating paths for each cost group. It also gives the instance count, total area (net and cell area), cell area, leakage power, dynamic power, runtime, and host name information.

If you perform physical synthesis and start with a floorplan, the report also contains the floorplan utilization in %. If you executed the `synthesize -to_placed` command, the report will also contain a `Silicon Virtual Prototype` section that lists the total and average net length in micron, and the routing congestion in %. Routing congestion is a measure of track overflow. A value greater than 5% in either direction gives an indication that the design will be difficult to route. The information is static information from the most recent **Encounter®** batch job.

Note: In case of a multi-mode design, the TNS column is not printed.

Options and Arguments

<i>design</i>	Specifies the design name on which to report. If no design is specified, the report is given for the current design.
<i>file</i>	Specifies the file to which the report must be written.
<code>-levels_of_logic</code>	Prints the number of gates on the critical path per cost group.
<code>-nopower</code>	Suppresses the power information in the report.

Examples

- The following example reports the QoR data for the current design.

```
rc:\> report qor
=====
...
Module:          cscan
Technology libraries: tutorial 1.0
                  slow_hvt 1.1
Operating conditions: typical_case (balanced_tree)
Wireload mode:    enclosed
=====
Timing
-----
Cost   Critical      Violating
Group Path Slack TNS     Paths
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```

-----
I2C      1182.4  0    0
C2O      1987.2  0    0
.....
Instance Count
-----
Leaf Instance Count      41
Sequential Instance Count 16
Combinational Instance Count 25
Hierarchical Instance Count 0

Area & Power
-----
Total Area                106.445
Cell Area                 106.445
Leakage Power              0.583 nW
Dynamic Power              8714.675 nW
Total Power                8715.259 nW

Max Fanout                2 (out1[0])
Min Fanout                1 (in2[3])
Average Fanout             1.2
Terms to net ratio          0.5
Terms to instance ratio     3.0
Runtime                     4.19 seconds
Hostname                   rcae030.cadence.com

```

- The following command requests to report the number of gates on the critical path per cost group data. This adds a No of gates on Critical Path column the table under Timing. The rest of the report does not change.

```

rc:\> report qor -levels_of_logic
=====
...
=====

Timing
-----
Cost   Critical      No of gates on Violating
Group  Path Slack TNS  Critical Path Paths
-----
I2C      1182.4  0    2    0
C2O      1987.2  0    1    0
.....

```

- The following example reports the QoR data for a Dynamic Voltage Frequency Scaling (DVFS) design (design with multiple power modes). In this case, an additional Mode column is added to the table under Timing. The rest of the report does not change.

```

Timing
-----
Mode   Cost   Critical Path Violating
      Group   Slack   Paths
-----
m1    default      No paths
      I2C        -202.6    1
      C2O        -116.5    1
      C2C        No paths
      I2O        No paths
m2    default      -202.6    2

```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
I2C          No paths  
C2O          No paths  
C2C          No paths
```

```
...
```

- The following example reports the QoR data after you executed the `synthesize -to_placed` command.

```
rc:/> report qor  
=====  
...  
Module:           fifo  
Technology libraries: slow 1.3  
                           physical_cells  
Operating conditions: slow  
Interconnect mode:   ple1  
Area mode:         physical library  
=====  
  
Timing  
-----  


| Cost Group | Critical Path | Slack  | TNS    | Violating Paths |
|------------|---------------|--------|--------|-----------------|
| default    | No paths      |        | 0      |                 |
| CLK1       |               | -802.8 | -47540 | 90              |
| Total      |               |        | -47540 | 90              |

  
Instance Count  
-----  


|                              |     |
|------------------------------|-----|
| Leaf Instance Count          | 209 |
| Sequential Instance Count    | 82  |
| Combinational Instance Count | 127 |
| Hierarchical Instance Count  | 0   |

  
Area & Power  
-----  


|                              |                |
|------------------------------|----------------|
| Total Area                   | 6744.198       |
| Cell Area                    | 5195.741       |
| <b>Floorplan Utilization</b> | <b>59.05%</b>  |
| Leakage Power                | 0.134 nW       |
| Dynamic Power                | 1112757.869 nW |
| Total Power                  | 1112758.004 nW |


|                         |              |
|-------------------------|--------------|
| Max Fanout              | 82 (n_60)    |
| Min Fanout              | 1 (d)        |
| Average Fanout          | 3.4          |
| Terms to net ratio      | 4.6          |
| Terms to instance ratio | 5.0          |
| Runtime                 | 7.66 seconds |
| Hostname                | rcae003      |

  
Silicon Virtual Prototype  
-----  


|                    |                   |
|--------------------|-------------------|
| Total Net Length   | 7280.40 um        |
| Average Net Length | 32.65 um          |
| Routing Congestion | H: 0.00% V: 0.00% |


```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

Affected by this attribute: [disable_power_mode](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report scan_compressibility

```
report scan_compressibility -directory atpg_directory [> file]
```

Reports the scan compressibility of a design.

Options and Arguments

-directory atpg_directory

Specifies the directory containing the ATPG compression runs created by the analyze_scan_compressibility command.

Default: current_working_directory/asc

file

Specifies the name of the file to which the report must be written.

Examples

- The following command reports the compressibility analysis resulting from the ATPG directory asc0.

```
rc:>report scan_compressibility -directory asc0
```

Analyzing from work directory 'asco'

```
-----  
Results of analyze_scan_compressibility
```

```
-----  
Design      - test  
Decompressor - broadcast  
Compressor   - missr  
mask        - widel
```

```
Achieved compression table with fullscan topup vectors
```

IC	TATR	TDVR	ATCov.	CL	Pat-comp	Pat-fs	Cycles	Runtime	Gatecount	Area
fs	1.0	1.0	100.00%	63	-	3	336	00:00:00	-	-
2	0.3	0.5	99.91%	32	9	5	1258	00:00:01	437	24774.3999999

```
Achieved compression table without fullscan topup vectors
```

IC	TATR	TDVR	ATCov.	CL	Pat-comp	Pat-fs	Cycles
fs	1.0	1.0	100.00%	63	-	3	336
2	0.4	1.2	98.76%	32	9	-	838

Total atpg runtime for exp. 00:00:01 hrs.

```
IC      - Inserted compression  
TATR    - Test application time reduction  
TDVR    - Test data volume reduction  
Cov.    - Atpg coverage  
CL      - Channel Length  
Pat-comp - No. of compression test patterns  
Pat-fs   - No. of fullscan test patterns  
Runtime  - Atpg runtime  
fs       - fullscan run
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

[Analyzing and Reporting Scan Compressibility](#) in *Design for Test in Encounter RTL Compiler*.

Affected by this command:

[analyze scan compressibility](#) on page 574

report sequential

```
report sequential
  [ [-instance_hier instance] [-hier] | -delete_seqs ]
  [subdesign | design] [> file]
```

Generates a report on the sequential elements of the current design. The report provides the sequential element name, row, column, filename information, and the sequential element type (flip-flop (async set/rest, sync set/reset, sync enable), latch, or timing model).

Note: Set the `hdl_track_filename_row_col` attribute to `true` before using the `elaborate` command to track the filename, row and column information.

Options and Arguments

<i>file</i>	Specifies the name of the file to which to write the report.
-delete_seqs	Reports the sequential elements that were removed during optimization. For example, sequential elements can be removed during constant propagation, redundancy removal, when merged with other sequential instances, or when they are not driving any primary outputs. Note: If the <code>delete_unloaded_seqs</code> attribute is set to <code>false</code> , the sequential elements that were optimized but did not get deleted will also be reported. The reason for these flops is reported with an asterisk (*). In this case, the flop's output pin will be dangling, not driving any loads. The loads driven by the flop before optimization, will be driven by a constant, or by the output pins of the merged flops pin. Note: This option cannot be combined with any other option.
-hier	Reports all the flops in the design.
-instance_hier <i>instance</i>	Specifies the hierarchical instance name to report flops.
<i>subdesign</i> <i>design</i>	Specifies the design or subdesign name to report flops.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following example reports all the sequential elements in the top level design:

```
rc:/> report sequential
=====
Generated by:          Version
Generated on:          Date
Module:                test
Technology library:   slow 1.5
Operating conditions: slow (balanced_tree)
Wireload mode:         segmented
=====
                                         Instantiated/
Register      File  Row Column  Inferred           Type
-----
sync_rst_reg  all.v 12   16    inferred  flip-flop synchronous reset
async_rst_reg all.v 21   27    inferred  flip-flop asynchronous reset
sync_set_reg  all.v 30   37    inferred  flip-flop asynchronous set
no_rst_reg    all.v 39   45    inferred  flip-flop
sync_preset_reg all.v 44   58    inferred  flip-flop synchronous set
q_reg        all.v 6    12    inferred  latch
```

- The following example reports all the sequential elements in the design:

```
rc:/> report sequential -hier
report sequential: prints a sequential instance report.
=====
...
=====
                                         Instantiated/
Register      File  Row Column  Inferred           Type
-----
m2/m3/m4/m5/o_m5_0_reg_1 hier.v 25   22  instantiated  flip-flop synchronous
                                         enable
m2/m3/m4/m5/o_m5_0_reg_2 hier.v 27   22  instantiated  flip-flop synchronous
                                         enable
m2/m3/m4/m5/o_m5_1_reg_0 hier.v 30   22  instantiated  flip-flop synchronous
                                         enable
m2/m3/m4/m5/o_m5_0_reg_0 hier.v 23   22  instantiated  flip-flop synchronous
                                         enable
.....
.....
m2/o_m2_clk1_0_reg_2      hier.v 174  27  instantiated  flip-flop synchronous
                                         enable
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports a timing model:

```
rc:/> report sequential
=====
...
=====
          Instantiated/
Register      File      Row Column   Inferred      Type
-----
clockgate/g2clatch_tlat timing_model.v  22   29   instantiated  timing_model
```

- The following example reports the sequential elements that were removed during optimization:

```
rc:/> report sequential -deleted_seqs
=====
...
=====
      Reason      Instance Name
-----
merged      ctl_wr_reg[36] merged with ctl_wr_reg[0]
merged      ctl_wr_reg[46] merged with ctl_wr_reg[10]
merged      ctl_wr_reg[47] merged with ctl_wr_reg[11]
merged      ctl_wr_reg[40] merged with ctl_wr_reg[12]
merged      ctl_wr_reg[41] merged with ctl_wr_reg[13]
...
...
merged      ctl_wr_reg[3] merged with ctl_wr_reg[39]
merged      ctl_wr_reg[8] merged with ctl_wr_reg[44]
merged      ctl_wr_reg[9] merged with ctl_wr_reg[45]
unloaded    ctl_wr_reg[32]
unloaded    ctl_wr_reg[12]
unloaded    ctl_wr_reg[20]
unloaded    ctl_wr_reg[28]
```

If the `delete_unloaded_seqs` root attribute is set to false, the report may look like:

```
rc:/> report sequential -deleted_seqs
=====
...
=====
      Reason      Instance Name
-----
merged (*) ctl_wr_reg[ 36 ] merged with ctl_wr_reg[ 0 ]
merged (*) ctl_wr_reg[ 46 ] merged with ctl_wr_reg[ 10 ]
merged (*) ctl_wr_reg[ 47 ] merged with ctl_wr_reg[ 11 ]
merged (*) ctl_wr_reg[ 40 ] merged with ctl_wr_reg[ 12 ]
merged (*) ctl_wr_reg[ 41 ] merged with ctl_wr_reg[ 13 ]
constant 0 (*) ctl_wr_reg[ 52 ]
constant 1 (*) ctl_wr_reg[ 53 ]
```

(*) indicates that the instance is optimized but not deleted
because root attribute 'delete_unloaded_seq' is set to false

report slew_calculation

```
report slew_calculation pin [-rise | -fall] [> file]
```

Reports how the slew of a cell driver pin is calculated from the look up table in the loaded technology library. The formula for calculating the delay is provided at the bottom of the report.

Options and Arguments

<i>file</i>	Redirects the report to the specified file.
[-fall -rise]	Uses the falling or rising slew calculation on the driver pin. By default, all possible arcs are reported.
<i>pin</i>	Specifies the cell driver pin.

report state_retention

```
report state_retention
  { [instance_list]
  | [-hierarchical] [-detail]
    [-power_gating_pin_driver {port|pin}...]
    [-power_domain power_domain-list]
  [-verbose] [> file]
```

Reports state retention information for the design. The return value of the report corresponds to the number of state-retention registers found.

Options and Arguments

<code>-detail</code>	Requests a detailed state retention report.
<code>file</code>	Specifies the name of the file to which to write the report.
<code>-hierarchical</code>	Reports state retention registers hierarchically. If this option is omitted, reports all state retention registers at the current level of the hierarchy.
<code>instance_list</code>	Reports detailed information for the specified state retention registers.
<code>-power_domain power_domain_list</code>	Reports all state retention registers in the specified power domains.
<code>-power_gating_pin_driver {pin port}</code>	Reports all state retention registers with the specified drivers.
<code>-verbose</code>	Lists the full paths of the state retention registers and power gating pins.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following command requests a hierarchical report. This report shows how many sequential instances are mapped to state retention registers and indicates why some sequential instances were not mapped.

```
rc:/> report state_retention -hier
=====
Generated by:          Encounter(R) RTL Compiler version
Generated on:          date
Module:                counter
Library domain:        umc_08v
Domain index:          0
Technology libraries: scmetropmk_umc13sp_tt_0p8v_25c 1.0
                      scmetropmk_umc13sp_tt_0p8v_1p2v_25c 1.0
                      scmetro_umc1130e_ll_tt_0p8v_25c 1.0
                      GS60_W_125_1.08_CORE_RET_SNPM.db
Operating conditions: nominal_(balanced_tree)
Library domain:        umc_10v
Domain index:          1
Technology libraries: scmetropmk_umc13sp_tt_1v_25c 1.0
                      scmetropmk_umc13sp_tt_0p8v_1v_25c 1.0
                      scmetro_umc1130e_ll_tt_1p0v_25c 1.0
                      GS60_W_125_1.08_CORE_RET_SNPM.db
Operating conditions: nominal_(balanced_tree)
Library domain:        umc_120v
Domain index:          2
Technology libraries: scmetropmk_umc13sp_tt_1p2v_25c 1.0
                      scmetropmk_umc13sp_tt_1v_1p2v_25c 1.0
                      scmetro_umc1130e_ll_tt_1p2v_25c 1.0
                      GS60_W_125_1.08_CORE_RET_SNPM.db
Operating conditions: nominal_(balanced_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====
```

```
Summary
-----
=====
Category           Number   %
=====
Number of Sequential instances mapped to SR cells      20    68.97
Number of Sequential instances not mapped to SR cells   9     31.03
-----
Total sequential instances in the design                 29.0   100
-----
```

Reason why not mapped to SR cell

```
-----
Excluded from mapping (no rule specified)            9    31.03
Did not find appropriate SR cell                   0     0.00
-----
```

Note: SR stands for State Retention

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the detailed report of the state retention registers in hierarchical instance `monitor_power`. It shows for each sequential instance that was replaced, the library cell that was used, to which power domain the instance belongs, and the names of the power gating pins. The report also indicates that all sequential instances in this hierarchical instance were remapped to state retention registers.

```
rc:/> report state_retention [find / -inst monitor_power]
...
=====
Module          State           Libcell      Power       Power      % of total
               Retention        Instance    Domain      Gating     seqs
=====
monitor_power/ cst_..._reg[0] umc_...F2_SNPM PDmon   1: (...NDRIVEN 27.59
                           2: (...~en[1])
                           cst_..._reg[1] umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_..._reg[2] umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_..._reg[3] umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_reg[0]   umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_reg[1]   umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_reg[2]   umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
                           cst_reg[3]   umc_...F2_SNPM PDmon   1: (...{en[0]}
                           2: (...~en[1])
=====
Summary
-----
=====
Category          Number      %
=====
Number of Sequential instances mapped to SR cells      8      27.59
Number of Sequential instances not mapped to SR cells  0      0.00
-----
Total sequential instances in the design                29.0     100
-----
Reason why not mapped to SR cell
-----
Excluded from mapping (no rule specified)            0      0.00
Did not find appropriate SR cell                   0      0.00
-----
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command shows the detailed hierarchical report for all state retention registers in power domain PDbin. The instance names, library cell names, and power gating pins are all abbreviated. To see the full names, add the -verbose option to the command.

```
rc:/> report state_retention -power_domain PDbin -hier -detail
...
=====
Module      State      Libcell      Power      Power      % of total
           Retention   Instance    Domain     Gating     seqs
=====
binary_counter/ bin_..._reg[0]  umc_..F2_SNPM  PDbin    1: (save) 0    13.79
                  bin_..._reg[1]  umc_..F2_SNPM  PDbin    1: (save) 0
                                         2: (...g1660/Y)
                  bin_..._reg[2]  umc_..F2_SNPM  PDbin    1: (save) 0
                                         2: (...g1660/Y)
                  bin_..._reg[3]  umc_..F2_SNPM  PDbin    1: (save) 0
                                         2: (...g1660/Y)
=====
Summary
-----
=====
Category          Number      %
=====
Number of Sequential instances mapped to SR cells      4      13.79
Number of Sequential instances not mapped to SR cells    1      3.45
Total sequential instances in the design                29.0     100
-----
Reason why not mapped to SR cell
-----
Excluded from mapping (no rule specified)            1      3.45
Did not find appropriate SR cell                   0      0.00
-----
Note: SR stands for State Retention
~ stands for active low signal
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command reports all state retention instances with the specified power gating pin.

```
report state_retention \
-power_gating_pin_driver [find / -pin monitor_power/g1660/Y ] -hier -detail
...
=====
Module          State           Libcell        Power       Power      % of total
Retention      Instance        Domain        Gating     Pins      seqs
=====
binary_counter/ bin_..._reg[0] umc_...F2_SNPM PDbin      1: (save) 0    13.79
                  bin_..._reg[1] umc_...F2_SNPM PDbin      2: (...g1660/Y
                  bin_..._reg[2] umc_...F2_SNPM PDbin      1: (save) 0    2: (...g1660/Y
                  bin_..._reg[3] umc_...F2_SNPM PDbin      1: (save) 0    2: (...g1660/Y
=====
Summary
-----
=====
Category          Number      %
=====
Number of Sequential instances mapped to SR cells      4      13.79
Number of Sequential instances not mapped to SR cells   9      31.03
-----
Total sequential instances in the design                29.0     100
-----
Reason why not mapped to SR cell
-----
Excluded from mapping (no rule specified)            9      31.03
Did not find appropriate SR cell                   0      0.00
-----
Note: SR stands for State Retention
~ stands for active low signal
```

4

Related Information

Affected by these commands: [commit_cpf](#) on page 914
 [read_cpf](#) on page 922

report summary

```
report summary  
  [-mode mode] [-all]  
  [design]... [> file]
```

Reports the area by mode used by the design, cells mapped for the blocks in the specified design, the wireload model, and the timing slack of the critical path. It also reports if any design rule is violated and the worst violator information.

Options and Arguments

-all	Reports all timing and DRC violations in the design.
<i>design</i>	Specifies the design for which you want to generate a report. By default, a report is created for all designs currently loaded in memory.
<i>file</i>	Specifies the name of the file to which to write the report.
-mode <i>mode</i>	Specifies the mode for which the report must be specified. Note: This option is only required for a design using a dynamic voltage frequency scaling methodology.

Command Reference for Encounter RTL Compiler

Analysis and Report

Example

- The following example generates a report of the area used by the design, the worst timing endpoint in the design, and the design rule violations summary.

```
rc:/> report summary
=====
Generated by:          RTL Compiler (RC) version
Generated on:          date
Module:                alu
Technology library:   tutorial 1.0
Operating conditions: typical case (balanced_tree)
Wireload mode:         enclosed
=====

Timing
-----
Slack      Endpoint
-----
-1082ps out1_tmp_reg[9]/D

Area
-----
Instance      Cells    Cell Area    Net Area    Wireload
-----
gen_test      326        525          0          AL_MEDIUM (S)
(S) = wireload was automatically selected

Design Rule Check
-----
Max_transition design rule: no violations.
Max_capacitance design rule (violation total = 19402.5)
Worst violator:
Pin           Load (ff)      Max       Violation
-----
in0[5] (Primary Input)     96.9       5.0       91.9
Max_fanout design rule (violation total = 16.000)
Worst violator:
Pin           Fanout      Max       Violation
-----
in0[5] (Primary Input)     8.000      4.000      4.000
```

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*

- [Generating a Summary Report](#)
- [Performing Multi-Mode Timing Analysis](#)

Affected by this command: [synthesize](#) on page 348
 [create_mode](#) on page 287

report test_power

```
report test_power
  [ [-clock float] [-flop_toggle_percentage float]
  | -atpg -library string [-clock float]
    [-directory path] [-capture | -scan_shift]
    [-lower_bound | -upper_bound]
    [-compression_mode mode]
  | -tcf file]
  [-power_mode {mode|power_mode}] [> file]
```

Reports the estimated average power consumption of the design during test. Additionally, when using the `-atpg` option, the command reports the average switching activities for the scan shift or capture mode. The test power will be estimated using the power tables in the synthesis library (.lib).

Note: For more information on the exact product requirements needed to use this command, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-atpg` Invokes Encounter Test to compute the toggle activity from the test patterns.

Note: To use this option, you must first build the scan chains using the `connect_scan_chains` command. Or if you start from a netlist with existing scan chains, you must first analyze the scan chains using the `define_dft_scan_chain -analyze` command before you can use this option.

`-capture` Estimates the average power consumed during the capture mode of the test patterns.

If neither the `-scan_shift` or `-capture` option are specified, the default is `-scan_shift`.

`-clock float` Specifies the frequency in MHz for the specified test mode (capture or scan).

Default: frequency of the test clock with highest frequency.

Command Reference for Encounter RTL Compiler

Analysis and Report

-compression_mode {COMPRESSION | COMPRESSION_DECOMP | OPMISRPLUS | OPMISRPLUS_DECOMP}

Specifies which compression test mode to select for power analysis.

If you omit this option, the default test mode that will be used is FULLSCAN.

-directory string Specifies the work directory to which the script files must be written. If the script files exist, the command can overwrite them.

Default: current_working_directory/test_power
file Specifies the file to which to redirect the report.

-flop_toggle_percentage float

Specifies the percentage of the flops that are changing state in each test clock cycle.

Default: 50% of the activity level in the design.

-library string Specifies the list of Verilog structural library files. Specify the list in a quoted string.

Note: This option is only required when you invoke this command on a mapped netlist.

-lower_bound Runs ATPG with repeat fill to provide a lower bound on the estimated power.

If you specified neither the **-lower_bound** nor the **-upper_bound** option, the **-upper_bound** option will be used by default.

Note: This option must be specified with the **-atpg** option.

-power_mode {mode|power_mode}

Specifies the mode for which the test power must be reported. You can either specify a mode or power mode name.

Note: This option is required when the design has multiple power modes.

-scan_shift Reports the average power consumed in scan shift mode.

If neither the **-scan_shift** or **-capture** option are specified, the default is **-scan_shift**.

Command Reference for Encounter RTL Compiler

Analysis and Report

<code>-tcf file</code>	Specifies the TCF file from which to read the switching activities.
<code>-upper_bound</code>	Runs ATPG with random fill to provide an upper bound on the estimated power. If you specified neither the <code>-lower_bound</code> nor the <code>-upper_bound</code> option, the <code>-upper_bound</code> option will be used by default.
	Note: This option must be specified with the <code>-atpg</code> option.

Examples

- The following three sets of commands are equivalent. They all specify two files to be used as Verilog simulation libraries.

```
set simLibs "sim/tsmc13.v sim/tpz013g3.v"
report test_power -atpg -library $simLibs

set rootDir sim
set verilogLibs "$rootDir/tsmc13.v $rootDir/tpz013g3.v"
report test_power -atpg -library $verilogLibs

set rootDir sim
report test_power -atpg -library "${rootDir}/tsmc13.v ${rootDir}/tpz013g3.v"
```

- The following command requests to estimate the test power consumed in scan shift mode when a test clock frequency of 20 MHz is applied, using the default flop toggle percentage of 50%.

```
rc:/> report test_power -clock 20
Computing the test power with the following toggle frequencies:
... set clocks @ 20 MHz
... set flops @ 50%
=====
Generated by:           version
Generated on:          date
Module:                cpu
Technology library:    typical 1.3
Operating conditions: typical (balanced_tree)
Wireload mode:         segmented
Area mode:             timing library
=====

      Leakage   Internal     Net     Dynamic      Total
                           (Internal+Net) (Leakage+Dynamic)
Instance Cells Power(nW) Power(nW) Power(nW) Power(nW) Power(nW)
-----cpu       1398     0.417  66742.159 120719.340 187461.499    187461.916
```

- The following command requests to compute the upper bounds of the test power in scan shift mode from the ATPG scan patterns with a test clock frequency of 20 MHz. This command requests to use the FULLSCAN scan structure (default).

```
report test_power -atpg -clock 20 -upper_bound -directory rc_rtp_UB \
-library $simLibs
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command requests to compute the lower bounds of the test power in scan shift mode from the ATPG scan patterns with a test clock frequency of 20 MHZ. This command requests to use the FULLSCAN scan structure (default).

```
report test_power -atpg -clock 20 -lower_bound -directory rc_rtp_LB \
-library $simLibs
```

- The following command requests to compute the test power in capture mode from the ATPG scan patterns with a test clock frequency of 20 MHZ

```
report test_power -atpg -clock 20 -capture -directory rc_rtp_CAP \
-library $simLibs
```

- The following command requests to compute the test power from the specified TCF file.

```
report test_power -tcf top.tcf
```

Related Information

[Analyzing the Test Power in Design for Test in Encounter RTL Compiler](#)

Affected by this constraint:

[define_dft test_clock](#) on page 693

Command Reference for Encounter RTL Compiler

Analysis and Report

report timing

```
report timing [-endpoints] [-summary] [-lint]
  [-full_pin_names] [-physical] [-num_paths integer]
  [-slack_limit integer] [-worst integer]
  [-from {instance|external_delay|clock|port|pin}...]
  [-through {instance|port|pin}...]...
  [-to {instance|external_delay|clock|port|pin}...]
  [-paths string] [-exceptions exception...]
  [-cost_group cost_group] [-mode mode_name] [-gtd] [-gui] [> file]
```

Generates a timing report of the current design. By default, the report gives a detailed view of the critical path of the current design. If the current session has multiple designs, use the `cd` command to navigate to the desired design to generate the report. You can also generate a report on possible timing constraint problems (timing lint) or view slack at endpoints.

Note: All values are always expressed in picoseconds. This unit cannot be changed.



When RTL Compiler detects a combinational feedback loop, it inserts a buffer from the technology library as a loop breaker instance before it performs timing analysis. To add the loop breaker, RTL Compiler might first need to uniquify a hierarchical instance which can result in a change in the netlist. As the module and instance name can change, this can affect your scripts and your database search.

Where applicable, special footnotes are used as shown in the table below to enhance the usability of the report.

Footnote	Meaning
(*)	Zero Slack Borrow Limit = This is the maximum amount that can be borrowed without violating timing on the lending side
(@)	Annotated capacitance
(a)	Net has asynchronous load pins which are being considered ideal
(b)	Timing paths are broken
(C)	Cell belongs to congested region (applies only to physical flow)
(i)	Net is ideal
(m)	Attribute <code>cell_delay_multiplier</code> is modified for this library cell
(P)	Instance is preserved
(p)	Instance is preserved but may be resized
(u)	Net has unmapped pin(s)
(V)	Net with virtual buffer(s)

Command Reference for Encounter RTL Compiler

Analysis and Report

Options and Arguments

<code>-cost_group <i>cost_group</i></code>	Reports only paths for the specified cost groups.
<code>file</code>	Specifies the name of the file to which to write the report.
<code>-endpoints</code>	Reports the slack at all timing endpoints in the design instead of the detailed path report. The most critical endpoints are listed first.
<code>-exceptions</code>	Reports only paths to which one of the specified exceptions applies. Note: This option can be combined with <code>-from</code> , <code>-through</code> , <code>-to</code> , and <code>-endpoints</code> options to further restrict the path reporting.
<code>-from {<i>instance external_delay clock port pin</i>}</code>	Specifies a Tcl list of start points for the paths. The start points can be input ports of your design, clock pins of flip-flops, clock objects, or a combination of these, instances, or input ports to which the specified external delay timing applies.
<code>-full_pin_names</code>	Prints the full hierarchical path of each pin in the report. You can use these pin names from the report to paste into other commands.
<code>-gtd</code>	Generates the MRTRF (Machine readable format) output out of RTL Compiler. This is a file format used for representing timing information that can be understood by the Global Timing Debug (GTD) viewer in Encounter. The GTD viewer in Encounter parses this file and represents the timing information in graphical format. This helps illustrate the total slack, failing paths, components contributing the most delay in each path, the total number of violations, and more in the design.
<code>-gui</code>	Allows you to create a detailed timing report in the GUI without having to use the <i>menu</i> commands. By using this option from the command line you can fine-tune the report using all command-line options which are not all available in the dialogs. Note: This option has only effect when you are running the tool in GUI mode.

Command Reference for Encounter RTL Compiler

Analysis and Report

-lint	Reports, in an abbreviated output, possible timing problems in the design. These problems can be caused by generated clocks, paths constrained with different clocks, ports that have no external delays, primary inputs that have no external driver or input transition set, primary outputs without external load, timing exceptions that cannot be satisfied, constraints that may have no impact on the design, and so on.
-mode <i>mode_name</i>	Reports the worst timing across all modes or analyzes timing in one particular mode.
-num_paths <i>integer</i>	Specifies the maximum number of paths to report. <i>Default:</i> the value of -worst Note: When combined with the -endpoints option, the number of endpoints is limited to the specified number.
-paths <i>string</i>	Reports only the specified timing restricted paths. Create the string argument using the <u>specify_paths</u> command.
-physical	Reports physical information, like the x, y location.
-slack_limit <i>integer</i>	Reports only paths with a slack smaller than the specified number.
-summary	Generates a short timing report that includes timing slack, start-point and end-point but does not include the full path.
-through { <i>instance</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of a sequence of points that a path must traverse. Points to traverse can be ports, hierarchical pins, pins on a sequential/mapped combinational cells, or sequential/mapped combinational instances. You can repeat the -through option to require that a path first must traverse one of the objects in the first set, then pass through one of the objects in the second set, and so on.
-to { <i>instance</i> <i>external_delay</i> <i>clock</i> <i>port</i> <i>pin</i> }	Specifies a Tcl list of endpoints for the paths. The endpoints can be output ports of your design, input pins of flip-flops, clock objects, or a combination of these, instances, or output ports to which the specified external delay timing exception applies.

Command Reference for Encounter RTL Compiler

Analysis and Report

Only paths that end at one of the ports or pins, or paths that are captured by one of the clock objects have the exception applied to them.

-worst integer Specifies the maximum number of paths to report to each endpoint.

Default: 1

Examples

- The following example generates a report for the worst path to each of the four most-constrained endpoints. The extraction of the report shows four different endpoints:

```
rc:/designs/sample_design> report timing -num_paths 4
=====
Generated by:          RTL Compiler (RC) version
...
=====
path    1:
      Pin           Type       Fanout   Load   Slew     Delay   Arrival
                           (ff)     (ps)    (ps)    (ps)    (ps)
-----
...
-----
Timing slack :      543ps
Start-point  : accum[1]
End-point    : aluout_reg_7/D

path    2:
      Pin           Type       Fanout   Load   Slew     Delay   Arrival
                           (ff)     (ps)    (ps)    (ps)    (ps)
-----
...
-----
Timing slack :      547ps
Start-point  : accum[1]
End-point    : aluout_reg_6/D

path    3:
      Pin           Type       Fanout   Load   Slew     Delay   Arrival
                           (ff)     (ps)    (ps)    (ps)    (ps)
-----
...
-----
Timing slack :     1030ps
Start-point  : accum[1]
End-point    : aluout_reg_5/D

path    4:
      Pin           Type       Fanout   Load   Slew     Delay   Arrival
                           (ff)     (ps)    (ps)    (ps)    (ps)
```

Command Reference for Encounter RTL Compiler

Analysis and Report

```
-----
...
-----
Timing slack : 1034ps
Start-point  : accum[1]
End-point    : aluout_reg_4/D
```

- The following example also generates a report for the four worst paths in the current design, but compared to the previous example, three of the worst paths now have the same endpoint.

```
rc:/designs/sample_design> report timing -num_paths 4 -worst 4
=====
```

```
Generated by:          RTL Compiler (RC) version
```

```
...
```

```
path 1:
```

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
-----	------	--------	--------------	--------------	---------------	-----------------

```
...
```

```
Timing slack : 543ps
```

```
Start-point  : accum[1]
```

```
End-point    : aluout_reg_7/D
```

```
path 2:
```

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
-----	------	--------	--------------	--------------	---------------	-----------------

```
...
```

```
Timing slack : 543ps
```

```
Start-point  : data[1]
```

```
End-point    : aluout_reg_7/D
```

```
path 3:
```

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
-----	------	--------	--------------	--------------	---------------	-----------------

```
...
```

```
Timing slack : 543ps
```

```
Start-point  : accum[1]
```

```
End-point    : aluout_reg_7/D
```

```
path 4:
```

Pin	Type	Fanout	Load (fF)	Slew (ps)	Delay (ps)	Arrival (ps)
-----	------	--------	--------------	--------------	---------------	-----------------

```
...
```

```
Timing slack : 547ps
```

```
Start-point  : accum[1]
```

```
End-point    : aluout_reg_6/D
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports the slack at the four most-constrained endpoints:

```
rc:/designs/sample_design> report timing -endpoints -num_paths 4
=====
Generated by:          RTL Compiler (RC) version
...
=====
Slack      Endpoint
-----
+543ps aluout_reg_7/D
+547ps aluout_reg_6/D
+1030ps aluout_reg_5/D
+1034ps aluout_reg_4/D
```

- The following example reports the path from input port 'a' that has the least slack:

```
rc:/> report timing -from [find / -port accum[1]]
=====
Generated by:          RTL Compiler (RC) version
...
=====
Pin           Type     Fanout   Load    Slew    Delay   Arrival
(fF)          (ps)     (ps)     (ps)
-----
(clock clock)      launch
(in del_1)        ext delay
alu7accum[1]       <<<  in port      6     66.5     0      +0      1000 R
                                         +1000    1000 F
                                         +0      1000 F
...
-----
Timing slack : 543ps
Start-point : accum[1]
End-point   : aluout_reg_7/D
```

- The following example reports the physical information:

```
rc:/> report timing -physical
=====
...
=====
Pin           Type     Fanout   Load    Slew    Delay   Arrival   Location
(fF)          (ps)     (ps)     (ps)     (ps)   (x,         y)
                               (R)
-----
(clock clock1)      launch
wr_addr_reg[0]/CK      launch
wr_addr_reg[0]/Q (@)  SDFFRHQX1      2   6.0   118   +311    311 R  (107640, 51660)
g154/A                (@)  CLKMX2X2      3   10.9   105   +206    517 R  (102580, 51660)
g146/B                (@)  ADDHXL        1   4.6    167   +189    706 R  (101660, 59040)
g145/A
...
-----
(clock clock1)      capture
                               10000 R
...
-----
Timing slack : 7997ps
Start-point : wr_addr_reg[0]/CK
End-point   : wr_addr_reg[7]/D
(@) : Annotated capacitance.
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following example reports only a condensed version of the timing report:

```
rc:/> report timing -summary
Timing slack : 9744ps
Start-point   : in1[3]
End-point     : out4
```

- The following example reports the path with the least slack that uses a `multi_cycle` exception named 'mc_2'

```
rc:/> report timing -exceptions [find / -exception mc_2]
```

- The following example reports only the worst path being launched by clock `clk1`

```
rc:/> report timing -paths [specify_paths -from clk1]
```

- The following example shows the summary printed at the end of the timing report when you use the `-lint` option. In this example 3 possible timing problems were found.

Lint summary	
Generated clocks without clock waveform	0
Generated clocks with incompatible options	0
Generated clocks with multi-master clock	0
Paths constrained with different clocks	0
Loop-breaking cells for combinational feedback	0
Nets with multiple drivers	0
Timing exceptions with no effect	0
Suspicious <code>multi_cycle</code> exceptions	0
Pins/ports with conflicting case constants	0
Inputs without clocked external delays	0
Outputs without clocked external delays	0
Inputs without external driver/transition	2
Outputs without external load	1
Exceptions with invalid timing start-/endpoints	0
Total:	3

- The following example illustrate how to use the `-gtd` option:

```
rc:/> report_timing -from ..... -to ..... -gtd > viol.mtarpt
```

Open `viol.mtarpt` in the global timing debug viewer in Encounter.

Related Information

See the following sections in *Setting Constraints and Performing Timing Analysis in Encounter RTL Compiler*:

- [Checking the Constraints Using the `report timing-lint` Command](#)
- [Performing Multi-Mode Timing Analysis](#)

Affected by this command: [create mode](#) on page 287
 [specify paths](#) on page 323
 [synthesize](#) on page 348

Command Reference for Encounter RTL Compiler

Analysis and Report

report units

```
report units  
[object] [ > file]
```

Reports the units used in the libraries.

You must have loaded the libraries before you can use this command.

Options and Arguments

<i>object</i>	Specifies the library for which you want to generate a report.
<i>file</i>	Redirects the report to the specified file.

Example

- In the following example two libraries, `slow` and `tutorial`, were loaded.

```
rc:/> report units  
Units  
-----  
LIBRARY          slow  
Time_unit        :1000ps  
Capacitive_load_unit :1000.0fF  
Resistance_unit   :1kohm  
Voltage_unit      :1V  
Current_unit       :1uA  
Power_unit         :nW  
  
LIBRARY          tutorial  
Time_unit        :1000ps  
Capacitive_load_unit :1000.0fF  
Resistance_unit   :1kohm  
Voltage_unit      :1V  
Current_unit       :1mA  
Power_unit         :nW
```

Related Information

Affects these commands: all report commands

Affected by this attribute: [lp_power_unit](#)

Command Reference for Encounter RTL Compiler

Analysis and Report

report utilization

```
report utilization [>file]
```

Reports the floorplan utilization for the design.

Options and Arguments

<i>file</i>	Redirects the report to the specified file.
-------------	---

Example

```
rc:/> report utilization
=====
Generated by:          Encounter(R) RTL Compiler 11.20
Generated on:         Mar 09 2012 06:11:47 pm
Module:               fifo
Technology libraries: slow 1.3
                      physical_cells
Operating conditions: slow
Interconnect mode:    global
Area mode:            physical library
=====
Total cell area (TCA) = 4399.660800000005
Total fixed cell area (TFA) = 0.0
Available row area (ARA) = 8614.305
utilization { {TCA - TFA}/ARA } = 0.51
```

Command Reference for Encounter RTL Compiler

Analysis and Report

report yield

```
report yield [-depth integer] [-min_count integer]  
[> file]
```

Reports the yield cost and yield percentage for each instance. This command is used in the design for manufacturing (DFM) flow.

Options and Arguments

-depth <i>integer</i>	Specifies the number of levels of recursion.
-min_count <i>integer</i>	Specifies the minimum instance count per block.
<i>file</i>	Redirects the report to the specified file.

Example

- The following example shows the defect-limited yield impact for library cell defects:

```
rc:/> report yield
```

Instance	Cells	Cell Area	Cost	Yield %
cpu	470	659	1.600606e-05	99.9984
alu1	248	283	7.606708e-06	99.9992
pcount1	65	92	2.215669e-06	99.9998
ireg1	33	88	1.629471e-06	99.9998
accum1	33	88	1.629471e-06	99.9998
decode1	50	67	1.568901e-06	99.9998

Related Information

[Design For Manufacturing Flow in Encounter RTL Compiler Synthesis Flows](#)

Affected by this command:	<u>read_dfm</u>
Related command:	<u>report gates -yield</u>
Affected by this attribute:	<u>optimize_yield</u>

Command Reference for Encounter RTL Compiler

Analysis and Report

statistics

```
statistics
  {add_metric | log | read | remove_metric | report
  | reset | run_stage_ids | write}
```

Tracks the statistics information (QoR metrics) for the design.

Options and Arguments

add_metric	Adds a user-defined metric to be tracked.
log	Computes the metrics at the stage where it is called and stores the data with the specified tag name.
read	Reads the metrics from the specified file into RTL Compiler.
remove_metric	Removes a user-defined metric.
report	Reports the metrics that were recorded at various stages or that were read in.
reset	Resets the metrics recorded or read in during the session.
run_stage_ids	Reports a summary of run IDs and the associated stage IDs and run description in the RC session.
write	Writes out the QoR metrics recorded at various stages to the specified database file

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Related commands:	statistics add_metric on page 509 statistics log on page 510 statistics read on page 512 statistics remove_metric on page 513 statistics report on page 514 statistics reset on page 517 statistics run_stage_ids on page 518 statistics write on page 519
-------------------	---

statistics add_metric

```
statistics add_metric
  -name metric -function function [argument]...
  [-header | -footer}
```

Adds a user-defined metric to be tracked.

Options and Arguments

-footer	Specifies to add the metric at the footer of the report. This allows to track static and dynamic values at the end of the report.
-header	Specifies to add the metric at the header of the report. This allows to track static values at the beginning of the report.
-name <i>metric</i>	Specifies the name of the user-defined metric.
-function <i>function</i>	Specifies the Tcl function to execute to compute the metric.
<i>argument</i>	Specifies an argument of the Tcl function

Example

- The following example defines a Tcl function `get_state` which returns the current design state. The user-defined metric is called `state`.

```
proc get_state {} {
    return [get_attr state /designs/cscan]
}
statistics add_metric -name state -function get_state
```

- The following example defines a metric to be added to the header of the report.

```
proc get_date {} {
    set $temp "[clock format [clock seconds] -format "%b%d-%T"]"
    return $temp
}
statistics add_metric -header -name Date -function get_date
```

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Affects this command: [statistics report](#) on page 514

Related command: [statistics remove_metric](#) on page 513

statistics log

```
statistics log  
  -stage_id string [-ignore_user_defined]
```

Computes the metrics at the stage where it is called and stores the data with the specified stage identifier (ID).

Options and Arguments

-ignore_user_defined

Specifies to ignore the user-defined metrics at this stage.

-stage_id *string*

Specifies the name of the user-defined stage.

To list the stage names already used during this run use the `statistics run_stage_ids` command.

Predefined stages are:

```
elaborate  
generic  
global_map  
incremental  
place  
incremental_place  
spatial  
spatial_incremental  
synthesize
```

Example

The following command computes the metrics after reading in the constraints and calls the `stage constraints`.

```
read_sdc my_constraints.sdc  
statistics log -stage_id constraints
```

Command Reference for Encounter RTL Compiler

Analysis and Report

Related Information

[Tracking and Saving QoR Metrics](#) in *Using Encounter RTL Compiler*

Affects these commands: [statistics report](#) on page 514

[statistics run_stage_ids](#) on page 518

Related command: [statistics run_stage_ids](#) on page 518

Command Reference for Encounter RTL Compiler

Analysis and Report

statistics read

```
statistics read  
-file file -reset
```

Reads the statistics information (QoR metrics) from the specified file into RTL Compiler.

You can read in multiple files, but you can only read in one file per command.

After reading in the information, the command returns similar info as the `statistics run_stage_ids` command.

Options and Arguments

<code>-file file</code>	Specifies the file to read the statistics information from.
<code>-reset</code>	Resets the existing content before reading the database file.

Example

```
rc:/> statistics read -file sample2.db  
Reading file sample2.db  
Sourcing './sample2.db' (Thu Aug 05 15:15:27 -0700 2010)...  
Done reading file sample2.db  
  
Run & Stage ID summary  
-----  
Run ID           Stage ID(s)           Run Description  
-----  
sample2 elaborate generic_global_map incremental n/a
```

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Affects this command:	<u>statistics run_stage_ids</u> on page 518
Related command:	<u>statistics write</u> on page 519
Related attributes:	<u>statistics run_id</u> <u>statistics run_description</u>

statistics remove_metric

```
statistics remove_metric  
  -name metric
```

Removes a previously user-defined metric.

Options and Arguments

-name *metric* Specifies the name of the user-defined metric to be removed.

Example

The following example removes the user-defined metric state.

```
statistics remove_metric -name state
```

Related Information

[Tracking and Saving QoR Metrics](#) in *Using Encounter RTL Compiler*

Affects this command: [statistics report](#) on page 514

Related command: [statistics add_metric](#) on page 509

statistics report

```
statistics report
  -run_id run_id [-compare run_id ]
  [-stage_id stage_tag [-compare_stage_id stage_tag]]
  [-ignore_user_defined] cp> file
```

Reports the statistics information (QoR metrics) that was recorded at various stages or that was read in, or compares the metrics of two specified runs.

Options and Arguments

- compare *run_id* Specifies the run that you want to compare to the run specified with the -run_id option.
- compare_stage_id *stage_tag*
 - Specifies the stage(s) of the run specified with the -compare option that you want to list in the report.
 - Note:** You cannot specify this option without the -stage_id option.
- ignore_user_defined
 - Specifies to ignore the user-defined metrics in the report.
- run_id *run_id* Specifies the run for which you want to see the statistics information.
- stage_id *string* Specifies the stage(s) of the run specified with the -run_id option that you want to list in the report.
 - If you omit this option, the report will show statistics information for all stages.

Command Reference for Encounter RTL Compiler

Analysis and Report

Examples

- The following command shows the report for run sample1 for the incremental stage.

```
statistics report -run_id sample1 -stage_id incremental
QOR statistics summary
-----
Metric          incremental
-----
WNS.I2O          no_value
WNS.I2C          1387.9
WNS.CLK1         2219.9
WNS.C2C          no_value
WNS.C2O          2187.2
WNS.default      no_value
TNS              0
Violating_paths  0
runtime          0.14
memory           0.00
Leakage_power    327.19
Net_power        362656.25
Internal_power   19369.41
Clock_gating_instances 0
total_net_length n/a
average_net_length n/a
routing_congestion n/a
utilization      0.0
Inverter_count   10
Buffer_count     0
timing_model_count 0
sequential_count 16
unresolved_count 0
logic_count      13
Total_area       538.01
Cell_area        538.01
Net_area         0.00
```

- The following report includes header and footer information defined with the `statistics add_metric -header` and `-footer` options.

```
QOR statistics summary
-----
Machine_Info rcae010 Intel(R) Xeon(TM) CPU 2.80GHz 2793.238Mhz 32bits 2_cores
Date Aug17-21:56:48
...
Metric          elaborate generic global_map incremental
-----
WNS.I2O          n/a      no_value  no_value  no_value
WNS.I2C          n/a      1533.5   1912.6   1387.9
WNS.CLK1         n/a      no_value  2219.9   2219.9
WNS.C2C          n/a      no_value  no_value  no_value
WNS.C2O          n/a      2187.2   2314.1   2187.2
WNS.default      no_value no_value  no_value  no_value
TNS              0         0         0         0
Violating_paths  0         0         0         0
...
Elapsed_Time 266
Super_Thread_Total_Runtime 245.000
```

Command Reference for Encounter RTL Compiler

Analysis and Report

- The following command does a comparison between two runs.

```
statistics report -run_id medium_effort -compare high_effort -stage global_map  
QoR statistics summary  
-----
```

Metric	global_map.medium_effort	global_map.high_effort	%diff
WNS.I2O	no_value	no_value	n/a
WNS.I2C	1912.6	1887.6	1.31
WNS.CLK1	2219.9	2219.9	0.0
WNS.C2C	no_value	no_value	n/a
WNS.C2O	2314.1	2314.1	0.0
WNS.default	no_value	no_value	n/a
TNS	0	0	n/a
Violating_paths	0	0	n/a
runtime	1.74	0.32	81.61
memory	8.05	1.11	86.21
Leakage_power	1730.88	1730.88	0.0
Net_power	453828.12	453828.12	0.0
Internal_power	59389.35	59389.35	0.0
Clock_gating_instances	0	0	n/a
total_net_length	n/a	n/a	n/a
average_net_length	n/a	n/a	n/a
routing_congestion	n/a	n/a	n/a
utilization	0.0	0.0	n/a
Inverter_count	10	10	0.0
Buffer_count	0	0	n/a
timing_model_count	0	0	n/a
sequential_count	16	16	0.0
unresolved_count	0	0	n/a
logic_count	13	13	0.0
Total_area	995.70	995.70	0.0
Cell_area	995.70	995.70	0.0
Net_area	0.00	0.00	n/a
state	mapped	mapped	n/a

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Affected by these commands: [statistics log](#) on page 510

[statistics read](#) on page 512

[statistics add_metric](#) on page 509

statistics reset

```
statistics reset
```

Removes the statistics information (QoR metrics) recorded or read in during the session.

Example

```
rc:/> statistics run_stage_ids
      Run & Stage ID summary
-----
Run ID          Stage ID(s)          Run Description
-----
sample2 elaborate generic global_map incremental n/a
rc:/> statistics reset
Info    : Reset the statistics information preset in the database. [STAT-4]
rc:/> statistics run_stage_ids
Info    : No run and stage_id data available to report. [STAT-12]
          : Either a statistics db file was not read in or the 'statistics_log_data'
attribute was not enabled for the synthesis run.
```

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Affects these commands:

[statistics report](#) on page 514

[statistics run_stage_ids](#) on page 518

statistics run_stage_ids

`statistics run_stage_ids`

Reports a summary of run IDs and the associated stage IDs and run description in the RC session.

The information is derived from the current run and from other database files that you read in.

Related Information

[Tracking and Saving QoR Metrics](#) in *Using Encounter RTL Compiler*

Affected by these commands: [statistics log](#) on page 510

[statistics read](#) on page 512

[statistics reset](#) on page 517

Related attributes: [statistics_run_id](#)

[statistics_run_description](#)

statistics write

```
statistics write  
[-to_file file]
```

Writes out the statistics information (QoR metrics) recorded at various stages to the specified database file.

Note: When you specify an existing file, the tool will overwrite the existing data.

Options and Arguments

-to_file file Specifies the file to which to write the statistics information.

If you omit this option, the data is recorded in the file determined by the `statistics_db_file` attribute.

Example

```
rc:/> set_attribute statistics_db_file stats/test.db /  
      Setting attribute of root '/': 'statistics_db_file' = stats/test.db  
rc:/> statistics write  
Info    : Writing statistics database to file. [STAT-3]  
        : Writing to db file 'stats/test.db'
```

Related Information

[Tracking and Saving QoR Metrics in Using Encounter RTL Compiler](#)

Related command: [statistics read](#) on page 512

Related attribute: [statistics_db_file](#)

timestat

```
timestat  
[string] [> file]
```

Reports the runtime and memory used up to this stage (time that the information was requested).

Options and Arguments

<i>string</i>	Specifies a user-defined string which allows you to identify at which stage in the design the information was requested. <i>Default:</i> undefined.
<i>file</i>	Redirects the command output to the specified file.

Examples

- The following script extract requests the information after RTL optimization.

```
synthesize -to_generic -eff $SYN_EFF  
puts "Runtime & Memory after 'synthesize -to_generic'"  
timestat GENERIC
```

- The following example shows the output after mapping.

```
rc:/> timestat map  
=====  
The RUNTIME after map is 0.45 secs  
and the MEMORY_USAGE after map is 24.16 MB  
=====
```

- The following example shows the output if no user-defined string was specified.

```
rc:/> timestat  
=====  
The RUNTIME after undefined is 0.45 secs  
and the MEMORY_USAGE after undefined is 24.16 MB  
=====
```

validate_timing

```
validate_timing
  [-sdc sdc_files] [-netlist path] [-libs lib_list]
  [-include file | -rep_tim_str command]
  [-keep_temp_dir] [> file]
```

Generates an Encounter Timing System timing report.

To run this command you need to have access to the Encounter® Timing System software.

Options and Arguments

<i>file</i>	Specifies the name of the file to which the report must be written. <i>Default:</i> vtim_ets_timing_rpt
<i>-include file</i>	Specifies the file containing the Encounter Timing System commands to be executed. These commands will be read in after the libraries, netlist and SDC constraints have been read into the Encounter Timing System tool.
<i>-keep_temp_dir</i>	Does not remove the temporary (.vtim_ets) directory in which the tool generates the SDC file or netlist.
<i>-libs lib_list</i>	Specifies the list of libraries to be read by Encounter Timing System. If no libraries are specified, the same libraries that were specified for synthesis are used.
<i>-netlist path</i>	Specifies the path to the netlist. If no netlist is specified, the netlist is generated with the write_hdl command in the .vtim_ets directory.
<i>-rep_tim_str command</i>	Specifies a string that contains a single Encounter Timing System report command.
<i>-sdc file_list</i>	Specifies the SDC file(s) for the design. If no SDC file(s) are specified, the SDC files are generated using the write_sdc command in the .vtim_ets directory.

Examples

- The following command reads the Encounter Timing System commands to be executed from the ets_include file.

```
validate_timing -netlist test.v -libs $env(REGLIBS)/tutorial.lib \
    -include ets_include
```

Because no SDC file was specified, the vtim_run_ets.sdc file is generated in the .vtim_ets directory.

- The following command directly specifies which Encounter Timing System command to be execute.

```
validate_timing -rep_tim_str "report_timing"
```

Physical

- [check_placement](#) on page 525
- [create_group](#) on page 527
- [create_placement_blockage](#) on page 528
- [create_placement_halo_blockage](#) on page 529
- [create_region](#) on page 530
- [create_row](#) on page 532
- [create_routing_blockage](#) on page 533
- [create_routing_halo_blockage](#) on page 534
- [create_track](#) on page 535
- [def_move](#) on page 536
- [generate_ple_model](#) on page 537
- [generate_reports](#) on page 539
- [move_blockage](#) on page 541
- [move_instance](#) on page 542
- [move_port](#) on page 543
- [move_region](#) on page 544
- [read_def](#) on page 545
- [read_encounter](#) on page 549
- [read_sdp_file](#) on page 550
- [read_spef](#) on page 551
- [report_congestion](#) on page 552

Command Reference for Encounter RTL Compiler

Physical

- [report utilization](#) on page 553
- [resize blockage](#) on page 554
- [resize region](#) on page 555
- [restore congestion map](#) on page 556
- [save congestion map](#) on page 557
- [specify floorplan](#) on page 558
- [summary table](#) on page 560
- [update congestion map](#) on page 562
- [update gcell congestion](#) on page 563
- [update gcell pin density](#) on page 564
- [update gcell utilization](#) on page 565
- [write def](#) on page 566
- [write sdp file](#) on page 567
- [write spef](#) on page 568

Command Reference for Encounter RTL Compiler

Physical

check_placement

```
check_placement design  
[-file file] [-verbose]
```

Checks the placement legality and highlights illegal objects. The command also returns some instance status statistics.

Note: This command applies only to the RC-P flow.

Options and Arguments

-file <i>file</i>	Specifies the file to which to write the report.
-verbose	Specifies to generate a verbose report.
<i>design</i>	Specifies the name of the design for which to create the group.

Examples

- The following example shows the non-verbose output of the `check_placement` command send to the console:

```
rc:/designs/DTMF_CHIP/> check_placement [find / -design DTMF*]  
Instance status statistics :  
    cover      : 0  
    fixed      : 71  
    placed     : 0  
    unplaced   : 5906  
Number of placement errors : 0  
  
Legality check : OK
```

- The following example shows an extract of the verbose output of the `check_placement` command:

```
rc:/designs/DTMF_CHIP/physical/rows> check_placement [find / -design DTMF*]  
-verbose  
Info      : Placement Information. [PLC-1]  
          : [I] Report Inst status stats...  
Info      : Placement Information. [PLC-1]  
          : [I] def / nl...0xd0d6b10 / 0xd0b3040  
Instance status statistics :  
    cover      : 0  
    fixed      : 71  
    placed     : 0  
    unplaced   : 5906  
Info      : Placement Information. [PLC-1]  
          : [I] Checking legality...  
Info      : Placement Information. [PLC-1]  
          : [I] check (127)  
Info      : Placement Information. [PLC-1]  
          : [I] checking overlaps...
```

Command Reference for Encounter RTL Compiler

Physical

```
Info      : Placement Information. [PLC-1]
           : [I] Obstruction rect (771280,297570 - 1097045,442015)
Info      : Placement Information. [PLC-1]
           : [I] Found 0 polygons and 1 rects.
Info      : Placement Information. [PLC-1]
           : [I] Obstruction rect (757845,512370 - 892440,1071790)
Info      : Placement Information. [PLC-1]
           : [I] Found 0 polygons and 1 rects.
....
Info      : Placement Information. [PLC-1]
           : [I] writing summary...
Number of placement errors : 0
```

create_group

```
create_group group
           instance_list
           [-region region]
           [-design design]
```

Creates a group of instances that must be placed close together.

The command returns the path to the `group` object that it creates. You can find the objects created by the `create_group` command in:

```
/designs/design/physical/groups/
```

The user-defined name is stored in the `def_name` group attribute.

Options and Arguments

<code>-design <i>design</i></code>	Specifies the name of the design for which to create the group.
<code>-region <i>name</i></code>	Associates the specified region with the group.
<code><i>group</i></code>	Specifies the name of the group to be created.
<code><i>instance_list</i></code>	Specifies the instances that belong to this group.

Example

The following example creates group `gr_reg` with two instances:

```
create_group mygroup -region myreg [find / -inst a_reg_3] [find / -inst a_reg_5]
```

Related Information

Affects this command: [synthesize](#) on page 348

Related command: [create_region](#) on page 530

create_placement_blockage

```
create_placement_blockage
    -boxes {llx lly urx ury}...
    [-component name]
    [-pushdown] [-partial] [-soft]
    [-density float] [-design design]
```

Creates a placement blockage.

Options and Arguments

-boxes {llx lly urx ury}...

Specifies the lower left and upper right coordinates of the blockage box. Use floats for the coordinates.

-component name

Associates the blockage with the specified component.

-density float

Specifies to percentage of the blockage area to be used for standard cells during initial placement.

-design name

Specifies the name of the design for which to create the blockage.

-partial

Creates a partial placement blockage.

-pushdown

Creates a pushdown placement blockage.

-soft

Creates a soft placement blockage.

Related Information

Affects this command:

[synthesize](#) on page 348

Related commands:

[create_placement_halo_blockage](#) on page 529

[move_blockage](#) on page 541

[resize_blockage](#) on page 554

create_placement_halo_blockage

```
create_placement_halo_blockage
    -left integer -right integer
    -top integer -bottom integer
    [-soft] instance
```

Creates a placement halo blockage around the specified instance.

Options and Arguments

<i>-bottom integer</i>	Specifies the halo width at the bottom of the instance. Specify the distance in DB units.
<i>-left integer</i>	Specifies the halo width at the left of the instance. Specify the distance in DB units.
<i>-right integer</i>	Specifies the halo width at the right of the instance. Specify the distance in DB units.
<i>-soft</i>	Creates a <u>soft</u> placement blockage.
<i>-top integer</i>	Specifies the halo width at the top of the instance. Specify the distance in DB units.
<i>instance</i>	Specifies the instance around which the placement halo is created.

Note: One DB unit is 1/1000 micron.

Related Information

Affects this command:	synthesize on page 348
Related commands:	create_placement_blockage on page 528 move_blockage on page 541 resize_blockage on page 554

create_region

```
create_region region
  [ -fence | -guide ] [-no_update]
  {-boxes {llx lly urx ury}... | -polygon pt pt pt [pt]...}
  [-design design]
```

Creates a region with the specified name.

The command returns the path to the `region` object that it creates. You can find the objects created by the `create_region` command in:

```
/designs/design/physical/regions/
```

The user-defined name is stored in the `def_name` region attribute.

Options and Arguments

`-boxes {llx lly urx ury}...`

Specifies the lower left and upper right coordinates of the box(es) that define the region. Use floats for the coordinates.

`-design design`

Specifies the name of the design for which to create the region.

`-fence`

Specifies to create a region of type `fence`.

`-guide`

Specifies to create a region of type `guide`.

`-no_update`

Prevents an update of the GUI when the new region is created.

`-polygon pt pt pt [pt]...`

Specifies a list of the coordinates of at least three points that define the region polygon. Use floats for the coordinates.

`region`

Specifies the name of the region to be created.

Example

The following command creates region `my_reg` of type `fence` for design DTMF.

```
create_region myreg -fence -box {200 200 1000 1000} [find / -design DTMF]
```

Related Information

Affects this command:

[synthesize](#) on page 348

Command Reference for Encounter RTL Compiler

Physical

Related commands: [move_region](#) on page 544

[resize_region](#) on page 555

Related Attributes: [Region Attributes](#)

Command Reference for Encounter RTL Compiler

Physical

create_row

```
create_row row
  -macro string
  -orientation {N|S|E|W|FN|FS|FE|FW}
  -llx integer -lly integer
  -height integer -width integer
  [-no_update] [-design design]
```

Creates a row with the specified name. Coordinates and dimensions must be specified in DB units, where one DB unit is 1/1000 micron.

The command returns the path to the *row* object that it creates. You can find the objects created by the `create_row` command in:

```
/designs/design/physical/rows/
```

Options and Arguments

<code>-design <i>design</i></code>	Specifies the name of the design for which to create the region.
<code>-height <i>integer</i></code>	Specifies the height of the row.
<code>-llx <i>integer</i></code>	Specifies the lower left X coordinate of the row.
<code>-lly <i>integer</i></code>	Specifies the lower left Y coordinate of the row.
<code>-macro <i>name</i></code>	Specifies the name of the LEF site to be used for the row.
<code>-no_update</code>	Prevents an update of the GUI when the new row is created.
<code>-orientation</code>	Specifies the orientation of the row to be created.
<code>-width <i>integer</i></code>	Specifies the width of the row.
<code><i>row</i></code>	Specifies the name of the row to be created.

Example

The following command creates row `my_row` for design DTMF.

```
create_row my_row -macro core_site -orientation N -llx 250 -lly 260 -width 900 \
-height 5 -design [find / -design DTMF]
```

Related Information

Affects this command: [synthesize](#) on page 348

Related Attributes: [Row Attributes](#)

create_routing_blockage

```
create_routing_blockage
  -layer layer
  -boxes {llx lly urx ury}...
  [-slots] [-fills] [-pushdown]
  [-component name] [-design <design>]
```

Creates a routing blockage on the specified layer.

Options and Arguments

-boxes {llx lly urx ury}...

Specifies the lower left and upper right coordinates of the blockage box. Use floats for the coordinates.

-component name

Associates the blockage with the specified component.

-design name

Specifies the name of the design for which to create the blockage.

-fills

Prevents metal fills on the specified layer in the specified areas.

-layer layer

Associates the blockage with the specified routing layer.

-pushdown

Specifies that the routing blockage must be pushed down from the top level.

-slots

Prevents slots on the specified layer in the specified areas.

Related Information

Affects this command:

[synthesize](#) on page 348

Related commands:

[create_routing_halo_blockage](#) on page 534

[move_blockage](#) on page 541

[resize_blockage](#) on page 554

create_routing_halo_blockage

```
create_routing_halo_blockage
    -min_layer layer -max_layer layer
    -distance integer
    instance
```

Creates a routing halo blockage around the specified instance.

Options and Arguments

<i>-distance integer</i>	Specifies the width of the halo around the specified instance. Specify the width in DB units
<i>instance</i>	Specifies the instance around which the routing halo is created.
<i>-min_layer layer</i>	Specifies the lowest routing layer for which to create the halo blockage.
<i>-max_layer layer</i>	Specifies the highest routing layer for which to create the halo blockage.

Note: One DB unit is 1/1000 micron.

Related Information

Affects this command: [synthesize](#) on page 348

Related command: [create_routing_blockage](#) on page 533

[move_blockage](#) on page 541

[resize_blockage](#) on page 554

create_track

```
create_track  
  -start integer -step integer -num integer  
  -layer string... [-vertical] [-design design]
```

Creates physical tracks (or routing grid) for the specified layer.

Options and Arguments

-design <i>design</i>	Specifies the name of the design in which the tracks must be created.
-layer <i>string</i>	Specifies the routing layer for which the tracks are created.
-num <i>integer</i>	Specifies the number of tracks to be created.
-start <i>integer</i>	Specifies the starting X or Y coordinate. X coordinate applies to vertical tracks, while the Y coordinate applies to horizontal tracks.
-step <i>integer</i>	Specifies the spacing between the tracks.
-vertical	Specifies to create vertical tracks. By default, a horizontal track is created.

Related Information

Related command: [synthesize](#) on page 348

def_move

```
def_move
  [-initialize]
  [-highlight]
  [-min_distance string]
```

Highlights cell movement in the physical tab of the GUI.

Use this command after you run the `synthesize -to placed` command.

Options and Arguments

<code>-highlight</code>	Highlights the cell movement with respect to their last stored location.
<code>-initialize</code>	Stores the location of the instances at the time the command is given with this option.
<code>-min_distance <i>string</i></code>	Limits the highlighting to cells that have been moved more than the specified distance. Specify the distance in microns. If this option is omitted, highlighting is limited to those cells whose x and y locations both changed by a value greater than or equal to the row height.

Example

Following illustrates the usage:

```
def_move -initialize
synthesize -to_placed -incr
def_move -highlight
```

Related Information

Related command: [synthesize](#) on page 348

generate_ple_model

```
generate_ple_model design -outfile file
```

Creates PLE correlation data for the design and stores this data in the specified file.

Net capacitances and resistances depend on technology parameters as well as the floorplan. This command refines the PLE parameters by taking both these variables into account and by comparing the PLE data with the SPEF data from Encounter. This results in a highly customized PLE equation for the given design and technology libraries.

The generated file is an encrypted file that contains

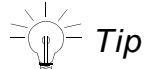
- Average Capacitance and Resistance values based on placement and default routing
- Adjustments for PLE equation parameters

The header of the generated file is readable. Check the header against the current design data to avoid miscorrelation. The header might look like:

```
# DESIGN NAME: DTMF_CHIP
# TECHNOLOGY LEF: all.lef
# CAP-TABLE: typical.captbl
# CAP SCALE: 1.0
# RES SCALE: 1.0
# ASPECT RATIO: 0.9814
```

If the design data is not inconsistent, the following message will be issued:

```
Warning : Inconsistent data. [PHYS-600]
          : Original design "DTMF_CHIP" not found in current session.
          : Input data used to create PLE correlation file is different from data
used in this session. This might lead to invalid results. Check design data.
```



Run this command once to generate PLE correlation data separately for each design. Source the generated file for every subsequent run that starts from RTL.

Note: This command does not require a floorplan, though having a good floorplan is highly recommended.

Options and Arguments

<i>design</i>	Specifies the design for which to generate the PLE data.
<i>-outfile file</i>	Specifies the name of the output file in which to store the PLE correlation data.

Command Reference for Encounter RTL Compiler

Physical

Example

In the beginning of the synthesis process, use the following flow to generate the PLE data:

```
set_attribute library library_list /
set_attribute lef_library lef_files /
read_hdl hdl_file
elaborate
read_def def_file
generate_ple_model design -outfile ple_file
```

In subsequent sessions use the following flow to load the PLE data:

```
set_attribute library library_list /
set_attribute lef_library lef_files /
read_hdl hdl_file
elaborate
decrypt ple_file
...
```

Related Information

Affects these commands: [synthesize -to_mapped](#)
 [synthesize -to_placed](#)

generate_reports

```
generate_reports -outdir path -tag string  
[-encounter]
```

Generates the QoS statistics at any stage in the flow, including timing, area, instance count, utilization and power.

Note: To create a summary table of the QoS statistics, use the `summary_table` command.

Options and Arguments

<code>-encounter</code>	Specifies to use the QoS statistics from the latest Encounter run. By default, the statistics are used from RTL Compiler.
<code>-outdir <i>path</i></code>	Specifies the path to the directory where the output data should be stored.
<code>-tag <i>string</i></code>	Specifies the tag name for reports generated at this stage.

Example

Assume the following `generate_reports` are executed.

```
rc:/> generate_reports -outdir TEST_OUT -tag initial  
...  
rc:/> generate_reports -outdir TEST_OUT -tag generic  
...  
rc:/> generate_reports -outdir TEST_OUT -tag map
```

Examining the `TEST_OUT` directory, you'll see the following reports were generated:

```
final.rpt  
generic_area.rpt  
generic_gates.rpt  
generic_qor.rpt  
generic_time.rpt  
incremental_area.rpt  
incremental_gates.rpt  
incremental_qor.rpt  
incremental_time.rpt  
initial_area.rpt  
initial_gates.rpt  
initial_qor.rpt  
initial_time.rpt  
map_area.rpt  
map_gates.rpt  
map_qor.rpt  
map_time.rpt
```

Command Reference for Encounter RTL Compiler

Physical

Related Information

Affects this command: [summary_table](#) on page 560

Affected by this attribute: [qos_report_power](#)

move_blockage

```
move_blockage blockage
  [-x integer | -dx integer]
  [-y integer | -dy integer]
```

Moves the specified blockage. You can specify either absolute coordinates, or a delta change in the coordinates. Coordinates or changes in coordinates must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-dx integer</i>	Specifies to move the blockage over the specified distance in the x-direction.
<i>-dy integer</i>	Specifies to move the blockage over the specified distance in the y-direction.
<i>-x integer</i>	Specifies the new x-coordinate for the lower left corner of the blockage.
<i>-y integer</i>	Specifies the new y-coordinate for the lower left corner of the blockage.
<i>blockage</i>	Specifies the name of the blockage to be moved.

Examples

- The following command moves blockage `my_blk` over 20 DB units in the vertical direction:

```
move_blockage -dy 20 [find . -blockage my_blk]
```

Related Information

Affects this command:	synthesize on page 348
Related commands:	create_placement_blockage on page 528 create_placement_halo_blockage on page 529 resize_blockage on page 554

move_instance

```
move_instance instance
    [-x integer | -dx integer]
    [-y integer | -dy integer]
```

Moves the specified instance. You can specify either absolute coordinates, or a delta change in the coordinates. Coordinates or changes in coordinates must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-dx integer</i>	Specifies to move the instance over the specified distance in the x-direction.
<i>-dy integer</i>	Specifies to move the instance over the specified distance in the y-direction.
<i>-x integer</i>	Specifies the new x-coordinate for the lower left corner.
<i>-y integer</i>	Specifies the new y-coordinate for the lower left corner.
<i>instance</i>	Specifies the name of the instance to be moved.

Examples

- The following command moves instance PLLCLK_INST over 2 DB units in the horizontal direction:

```
move_instance -dx 2 [find . -inst PLLCLK_INST]
```

move_port

```
move_port port
    [-x integer | -dx integer]
    [-y integer | -dy integer]
```

Moves the specified port. You can specify either absolute coordinates, or a delta change in the coordinates. Coordinates or changes in coordinates must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-dx integer</i>	Specifies to move the port over the specified distance in the x-direction.
<i>-dy integer</i>	Specifies to move the port over the specified distance in the y-direction.
<i>-x integer</i>	Specifies the new x-coordinate for the center of the port.
<i>-y integer</i>	Specifies the new y-coordinate for the center of the port.
<i>port</i>	Specifies the name of the port to be moved.

Examples

- The following command moves port vdd2 up over 200 DB units in the vertical direction:

```
move_port -dy 200 pllrst
```

move_region

```
move_region region
    [-x integer | -dx integer]
    [-y integer | -dy integer]
```

Moves the specified region. You can specify either absolute coordinates, or a delta change in the coordinates. Coordinates or changes in coordinates must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-dx integer</i>	Specifies to move the region over the specified distance in the x-direction.
<i>-dy integer</i>	Specifies to move the region over the specified distance in the y-direction.
<i>-x integer</i>	Specifies the new x-coordinate for the lower left corner of the region.
<i>-y integer</i>	Specifies the new y-coordinate for the lower left corner of the region.
<i>region</i>	Specifies the name of the region to be moved.

Examples

- The following command moves region `my_reg` over 20 DB units in the vertical direction:

```
move_region -dy 20 [find . -region my_reg]
```

Related Information

Affects this command:	synthesize on page 348
Related commands:	create_region on page 530 resize_region on page 555

Command Reference for Encounter RTL Compiler

Physical

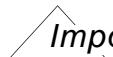
read_def

```
read_def
  [-design design]
  [-create_ports] [-power_switch_insert]
  [-no_nets] [-no_specialnets] [-no_vias]
  [-keep_filler_cells] [-keep_welltap_cells]
  [-hierarchical [-no_force_fixed] ]
  [-incremental] [-ignore_errors] def_file
```

Loads the specified DEF file.

RTL Compiler will perform a consistency check between the DEF and the Verilog netlist and issue relevant messages if necessary. The DEF file must define the die size. A warning message will be issued for any components that lie outside the die area. For better synthesis results, you should also have the pin and macro locations specified in the DEF, although it is not required.

RTL Compiler supports DEF 5.3 and above.



Important

The information extracted from the DEF file depends on the license you use to start the tool. In most cases, the `read_def` command will only extract the floorplan information (fixed macros, blockages, pins, regions, and so on). Detailed placement information (in particular, placed components) can only be extracted if you start the tool with a physical license.

Options and Arguments

<code>-create_ports</code>	Specifies to create an unconnected top-level netlist port for each pin in the DEF file that does not exist in the netlist and that is either of type clock, reset, scan, or signal.
<code>def_file</code>	Specifies the DEF file.
<code>-design design</code>	Specifies the design to which to annotate the DEF information.
<code>-hierarchical</code>	Specifies that the DEF file is hierarchical.
<code>-ignore_errors</code>	Ignores any inconsistency errors between the LEF and DEF.
<code>-incremental</code>	Specifies that the DEF file has incremental information. Therefore only updates of the pins and components are needed. <ul style="list-style-type: none">■ If a pin or component did not have physical data in the original DEF file, physical data in the incremental DEF are used.

Command Reference for Encounter RTL Compiler

Physical

- If a pin or component has physical data from the original DEF file, the physical data in the incremental DEF file will overwrite the original data.

-keep_filler_cells

Specifies to keep the physical-only filler cells. Filler or spacer cells are standard cells to fill in space between regular core cells.

-keep_welltap_cells

Specifies to keep the physical-only welltap cells. Welltap cells are standard cells that connect N and P diffusion wells to the correct power and ground wire.

-no_force_fixed

Prevents that components have their placement_status set to fixed in hierarchical mode (that is, when reading in a hierarchical DEF file).

-no_nets

Specifies to skip the NETS section in the DEF file.

-no_specialnets

Specifies to skip the SPECIALNETS section in the DEF file.

-no_vias

Specifies to skip the VIAS section in the DEF file.

-power_switch_insert

Inserts power switch cells in the netlist if they are present in the DEF file.

These components will be added to the existing netlist and will be connected.

Example

- The following example loads the point.def DEF file:

```
rc:/> read def point.def
Reading and processing DEF file 'point.def'...
Parsing DEF file...
Warning : A metal fill was found in the DEF file. [PHYS-178]
          : Metal fill on layer 'METALL'.
...
Warning : A DEF component does not exist in the netlist. [PHYS-171]
          : The component 'U1/fool' does not exist.
...
Info    : A COVER component has been read. [PHYS-182]
          : The instance 'ram_bank1' is COVER.
Info    : A COVER component has been read. [PHYS-182]
          : The instance 'U1/g20' is COVER.
...
Warning : Physical cell not created due to missing macro. [PHYS-211]
          : Macro 'FILL1MTR' for physical cell 'FILLER2' not found.
Done parsing DEF file.
```

Command Reference for Encounter RTL Compiler

Physical

```
Processing DEF file...
Warning : Overlapping guide detected. [PHYS-187]
          : Guide 'region_6' (DEF name: 'region_6') overlaps fence
          'region_2' (DEF name: 'REGION_TWO').
Warning : Overlapping guide detected. [PHYS-187]
          : Guide 'region_8' (DEF name: 'region_8') overlaps region
          'region_1' (DEF name: 'REGION_ONE').
Installing blockage router...
Summary report for DEF file 'point.def'

Components
-----
Cover: 3
Fixed: 10
Physical: 2
  Bump: 0
Placed: 591
Unplaced: 2
TOTAL: 608 (3 are class macro)
```

There are 2 components that do not exist in the netlist.

```
Pins
-----
Cover: 0
Fixed: 0
Physical: 0
Placed: 0
Unplaced: 0q
TOTAL: 0

Nets
-----
Read: 0 (cover: 0, fixed: 0)
Skipped: 0
TOTAL: 0

SpecialNets
-----
Read: 0
Skipped: 0
TOTAL: 0

Fences: 2
Guides: 5
Regions: 1

Done processing DEF file.
```

Physical Message Summary

```
2 / 2 W PHYS-171 Component not present in netlist.
2 / 2 W PHYS-178 Metal fill present.
13 / 0 I PHYS-181 Full preserve set on instance.
3 / 3 I PHYS-182 Cover component present.
2 / 2 W PHYS-187 Overlapping guide detected.
2 / 2 W PHYS-211 Physical cell not created due to missing macro.
2 / 2 W PHYS-214 Library cell not defined in physical library.
```

Done reading and processing DEF file (time: 0s).

Command Reference for Encounter RTL Compiler

Physical

Related Information

Affected by this attribute: [script_search_path](#)

Related attribute: [phys_ignore_special_nets](#)

read_encounter

```
read_encounter config configuration_file
```

Reads an Encounter configuration file into RTL Compiler. An Encounter configuration file is an ASCII file that contains Tcl variables that describe information such as the netlist or RTL, technology libraries, LEF information, constraints, and capacitance tables. Encounter configuration files have the .config extension.

After the file is loaded, the constraints and attributes specified in the configuration file will automatically be set. Hence, the design will be ready for synthesis or optimization or both.

Options and Arguments

configuration_file Specifies the configuration file to load.

Examples

- Since the configuration file contains information such as technology libraries, HDL files, and constraints, the `read_encounter` command should be used at the beginning of a synthesis session. After the configuration file is loaded, you can immediately synthesize or optimize the design. The following example loads the `fast.config` configuration file, then synthesizes the design to gates.

```
rc:/> read_encounter config fast.config  
rc:/> synthesize -to_mapped  
...
```

Related Information

Related command: [write_encounter](#) on page 243

read_sdp_file

```
read_sdp_file -file file  
  [-design design] [-preserve_size_ok]  
  [-hier_path path] [-origin llx lly]
```

Reads a relative placement file (in .sdp format) containing the SDP definitions.

Options and Arguments

<code>-design design</code>	Specifies the design in which to use the SDP information.
<code>-file file</code>	Specifies the name of the SDP input file.
<code>-hier_path path</code>	Specifies the hierarchical path name of the SDP elements specified in the SDP file. This path name is appended to all SDP instances defined in the file.
<code>-origin llx lly</code>	Specifies the coordinates of the origin of the SDP groups. You can use floats to specify the coordinates. All SDP groups use the same origin.
<code>-preserve_size_ok</code>	Sets the <code>preserve</code> attribute on the SDP instances to <code>preserve_size_ok</code> . <i>Default:</i> <code>preserve</code>

Example

The following command reads the `dtmpf_chip.sdp` file. It also appends the specified hierarchical path to all elements.

```
read_sdp_file -file dtmf_chip.sdp -hierPath DTMF_CHIP/TDSP_CORE_INST
```

Related Information

[SDP Flows in Design with RTL Compiler Physical](#)

Related command: [write_sdp_file](#) on page 567

read_spef

```
read_spef spef_file
           [-max_fanout integer]
           [-hierarchical] [-incremental]
```

Reads the SPEF file and loads the resistance and grounded capacitors from the file. Gzip compressed files (.gz extension) can also be loaded. In RTL Compiler, the SPEF file is generated by Encounter.

Options and Arguments

-hierarchical	Specifies that SPEF file is hierarchical.
-incremental	Specifies that the SPEF file contains incremental information. Allows to read in the data without resetting the original SPEF annotated values (if any).
-max_fanout	Any net with a fanout count greater than the specified value will not be annotated with the resistance and capacitance from the SPEF. Also, delay calculation will not be performed. The default value is 1000.
<i>spef_file</i>	Specifies the SPEF file.

Related Information

Related command: [write_spef on page 568](#)

report congestion

Refer to [report congestion](#) in [Chapter 8, “Analysis and Report.”](#)

report utilization

Refer to [report utilization](#) in [Chapter 8, “Analysis and Report.”](#)

resize_blockage

```
resize_blockage blockage
    [-left integer] [-right integer]
    [-top integer] [-bottom integer]
```

Resizes the specified blockage. All distances must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-bottom integer</i>	Specifies the distance over which the bottom edge of the blockage must be moved.
<i>-left integer</i>	Specifies the distance over which the left edge of the blockage must be moved.
<i>-right integer</i>	Specifies the distance over which the right edge of the blockage must be moved.
<i>-top integer</i>	Specifies the distance over which the top edge of the blockage must be moved.
<i>blockage</i>	Specifies the name of the blockage to be resized.

Example

The following command extends blockage `my_blk` with 20DB units on the right side.

```
resize_blockage my_blk -right 20
```

Related Information

Affects this command:	synthesize on page 348
Related commands:	create_placement_blockage on page 528 create_placement_halo_blockage on page 529 move_blockage on page 541

resize_region

```
resize_region region
    [-left integer] [-right integer]
    [-top integer] [-bottom integer]
```

Resizes the specified region. All distances must be specified in DB units, where one DB unit is 1/1000 micron. You must specify at least one option.

Options and Arguments

<i>-bottom integer</i>	Specifies the distance over which the bottom edge of the region must be moved.
<i>-left integer</i>	Specifies the distance over which the left edge of the region must be moved.
<i>-right integer</i>	Specifies the distance over which the right edge of the region must be moved.
<i>-top integer</i>	Specifies the distance over which the top edge of the region must be moved.
<i>region</i>	Specifies the name of the region to be resized.

Example

The following command extends region `my_reg` with 20DB units on the right side.

```
resize_region my_reg -right 20
```

Related Information

Affects this command:	synthesize on page 348
Related commands:	create_region on page 530 move_region on page 544

restore_congestion_map

```
restore_congestion_map [-design design] file
```

Restores the congestion map data from the specified file.

Note: This command applies only to the RC-P flow.

Options and Arguments

-design <i>design</i>	Specifies the design for which the congestion map must be restored.
<i>file</i>	Specifies the name of the file containing the congestion data.

Related Information

Related commands:	save congestion map on page 557
	update congestion map on page 562

save_congestion_map

`save_congestion_map [-design design] file`

Saves the congestion map to the specified file.

Note: This command applies only to the RC-P flow.

Options and Arguments

`-design design` Specifies the design for which the congestion map must be saved.

`file` Specifies the name of the file to which the congestion data must be saved.

Related Information

Related commands: [restore_congestion_map on page 556](#)
 [update_congestion_map on page 562](#)

specify_floorplan

```
specify_floorplan
  { -die_box {llx lly urx ury}
  | -die_points pt pt pt [pt]...
  | -height float -width float }
  [-core_box {llx lly urx ury} ]
  {design | subdesign}
```

Specifies the floorplan information for the specified design.

Options and Arguments

-core_box {llx lly urx ury}

Specifies the lower left and upper right coordinates of the core of the design. Specify the coordinates in micron.

{design | subdesign}

Specifies the name of the design or subdesign.

-die_box {llx lly urx ury}

Specifies the lower left and upper right coordinates of the die of the design. Specify the coordinates in micron.

-die_points pt pt pt [pt]...

Specifies a list of coordinates of at least three points if the die has a polygon geometry. Specify the coordinates in micron.

-height float

Specifies the height of the die. Specify the height in micron.

-width float

Specifies the width of the die. Specify the width in micron.

Examples

- The following commands first specify the width and height of the die, then the coordinates of the core. Parameter legality checking is performed as shown below.

```
rc:/> specify_floorplan DTMF_CHIP -height 1400 -width 1200
rc:/> specify_floorplan DTMF_CHIP -core_box {100 100 1400 1400}
Error   : The design core box must lie within the die box. [PHYS-102]
[specify_floorplan]
  : core box = {100 100 1400 1400}, die box = {0 0 1200 1400}
  : Wrong coordinates were specified for the core box.
```

- The following command specifies the coordinates of the die and the core of the design.

Command Reference for Encounter RTL Compiler

Physical

```
specify_floorplan MYCHIP -die_box {0 0 1550 1500} -core_box {100 100 1400 1400}
```

Related Information

Affects this command: [synthesize](#) on page 348

Command Reference for Encounter RTL Compiler

Physical

summary_table

```
summary_table -outdir path
```

Generates a summary table which includes various QoS numbers for various stages in the RC flow.

Note: You must have run the generate_reports before you use this command.

Options and Arguments

<code>-outdir <i>path</i></code>	Specifies the path to the directory where the output data of the this command must be stored.
----------------------------------	---

Example

The following report shows the summary of the statistics after three stages: initial stage, generic stage, and map stage, where initial, generic, and map were the tags specified in the subsequent generate_reports commands.

```
rc:/> summary_table -outdir $env(TEST_OUT_DIR)
```

```
Working Directory = /...test
QoS Summary for cscan
=====
Metric           initial   generic   map
=====
Slack (ps):      1958.6    1934.1    1190.1
R2R (ps):        no_value   no_value   no_value
I2R (ps):        1958.6    1934.1    1190.1
R2O (ps):        2152.6    2152.6    1987.2
I2O (ps):        no_value   no_value   no_value
CG (ps):         2100.0    2100.0    2100.0
TNS (ps):         0          0          0
R2R (ps):        no_value   no_value   no_value
I2R (ps):         0          0          0
R2O (ps):         0          0          0
I2O (ps):        no_value   no_value   no_value
CG (ps):         0          0          0
Failing Paths:   0          0          0
Area:            0          0          0
Instances:       36         28         43
Utilization (%): 0.00       0.00       0.00
Tot. Net Length (um): no_value   no_value   no_value
Avg. Net Length (um): no_value   no_value   no_value
Total Overflow H:  0          0          0
Total Overflow V:  0          0          0
Route Overflow H (%): no_value   no_value   no_value
Route Overflow V (%): no_value   no_value   no_value
=====
CPU Runtime (m:s): 00:04     00:00     00:01
Real Runtime (m:s): 00:13     01:35     01:30
```

Command Reference for Encounter RTL Compiler

Physical

```
CPU Elapsed (m:s):      00:09      00:10      00:12
Real Elapsed (m:s):     00:37      02:12      03:42
Memory (MB):            66.32      66.60      73.38
=====
```

```
=====
Flow Settings:
=====
```

```
Total Runtime (m:s): 03:25
Total Memory (MB):   73.38
Executable Version:  9.1
=====
```

Related Information

Affected by this command: [generate_reports](#) on page 539

Affected by this attribute: [qos_report_power](#)

update_congestion_map

```
update_congestion_map [-pin_density] [-design design]
```

Updates the congestion map for the specified design. You can run this command after any command that updates the physical data.

Note: This command applies only to the RC-P flow.

Options and Arguments

-design *design* Specifies the design for which the congestion map must be updated.

-pin_density Specifies to compute the pin density.

update_gcell_congestion

```
update_gcell_congestion [-design design]
```

Updates the congestion values of the gcells for the specified design. You can run this command after any command that updates the physical data.

Note: This command applies only to the RC-P flow.

Options and Arguments

-design <i>design</i>	Specifies the design for which the gcell congestion values must be updated.
------------------------------	---

update_gcell_pin_density

```
update_gcell_pin_density [-design design]
```

Updates the pin density values of the gcells for the specified design. You can run this command after any command that updates the physical data.

Note: This command applies only to the RC-P flow.

Options and Arguments

<code>-design <i>design</i></code>	Specifies the design for which the gcell pin density values must be updated.
------------------------------------	--

update_gcell_utilization

```
update_gcell_utilization [-design design]
```

Updates the utilization values of the gcells for the specified design. You can run this command after any command that updates the physical data.

Note: This command applies only to the RC-P flow.

Options and Arguments

<code>-design <i>design</i></code>	Specifies the design for which the gcell utilization values must be updated.
------------------------------------	--

Command Reference for Encounter RTL Compiler

Physical

write_def

```
write_def [-ignore_groups] [-ignore_placed_instances]
           [-scan_chains] [design] [> file]
```

Writes a floorplan, in DEF format, for the specified design. RTL Compiler does not store all the information from the original DEF (for example, VIAS, SLOTS, ROWS, TRACKS, etc.). However, the generated floorplan includes data from both the RTL Compiler session as well as the original imported DEF. The DEF does not contain the netlist information (net connectivity information) aside from the power/ground nets defined in the input DEF (SPECIALNETS section).

You can write out the DEF in gzip format by specifying the `.gz` extension when writing out the file.

Options and Arguments

<i>design</i>	Specifies a particular design for which to write out the floorplan. Only one design can be specified at a time.
<i>file</i>	Redirects the floorplan to the specified file.
<code>-ignore_groups</code>	Discards the instance groups that are defined in RC. These come from the input DEF (GROUPS section). The output DEF will have no GROUPS or REGIONS sections.
<code>-ignore_placed_instances</code>	Only writes preplaced instances (+ FIXED tag in the DEF) to the DEF. These are objects such as macros. Without this option, the output DEF will include all the instances that are preplaced or placed. Unplaced components will never be written to the DEF. This is all in the COMPONENTS section.
<code>-scan_chains</code>	Specifies that scan chain information should be included in the floorplan.

Related Information

Related attribute: [phys_ignore_special_nets](#)

write_sdp_file

```
write_sdp_file  
  [-file file]  
  [-top_group sdp_group]
```

Saves the current SDP relative placement information in the specified file.

Options and Arguments

- | | |
|-----------------------------|--|
| -file <i>file</i> | Specifies the name of the SDP output file.
<i>Default:</i> Output is written to the screen. |
| -top_group <i>sdp_group</i> | Specifies the SDP top group whose information to write out. |

Example

The following command writes the current SDP information in SDP format in file test.sdp.

```
write_sdp_file -file test
```

Related Information

[SDP Flows in Design with RTL Compiler Physical](#)

Related command: [read_sdp_file](#) on page 550

write_spef

`write_spef [> file]`

Writes out the parasitics (resistance and capacitance information) of the design in SPEF (Standard Parasitic Exchange Format) format.

Options and Arguments

file Specifies the file to which to write the parasitics.

Related Information

Related command: [read_spf](#) on page 551

Design for Test

- [add_opcg_hold_mux](#) on page 573
- [analyze_scan_compressibility](#) on page 574
- [analyze_testability](#) on page 583
- [check_atpg_rules](#) on page 586
- [check_dft_pad_configuration](#) on page 588
- [check_dft_rules](#) on page 589
- [check_mbist_rules](#) on page 595
- [compress_block_level_chains](#) on page 598
- [compress_scan_chains](#) on page 601
- [concat_scan_chains](#) on page 615
- [configure_pad_dft](#) on page 617
- [connect_compression_clocks](#) on page 618
- [connect_opcg_segments](#) on page 619
- [connect_scan_chains](#) on page 620
- [define_dft](#) on page 625
- [define_dft_abstract_segment](#) on page 628
- [define_dft_boundary_scan_segment](#) on page 633
- [define_dft_dft_configuration_mode](#) on page 636
- [define_dft_domain_macro_parameters](#) on page 639
- [define_dft_fixed_segment](#) on page 641
- [define_dft_floating_segment](#) on page 643

Command Reference for Encounter RTL Compiler

Design for Test

- [define_dft_jtag_instruction](#) on page 645
- [define_dft_jtag_instruction_register](#) on page 649
- [define_dft_jtag_macro](#) on page 651
- [define_dft_mbist_clock](#) on page 656
- [define_dft_mbist_direct_access](#) on page 659
- [define_dft_opcg_domain](#) on page 662
- [define_dft_opcg_mode](#) on page 665
- [define_dft_opcg_trigger](#) on page 667
- [define_dft_osc_source](#) on page 669
- [define_dft_preserved_segment](#) on page 671
- [define_dft_scan_chain](#) on page 674
- [define_dft_scan_clock_a](#) on page 680
- [define_dft_scan_clock_b](#) on page 683
- [define_dft_shift_enable](#) on page 686
- [define_dft_shift_register_segment](#) on page 689
- [define_dft_tap_port](#) on page 691
- [define_dft_test_clock](#) on page 693
- [define_dft_test_mode](#) on page 697
- [dft_trace_back](#) on page 701
- [fix_dft_violations](#) on page 703
- [fix_scan_path_inversions](#) on page 707
- [identify_multibit_cell_abstract_scan_segments](#) on page 708
- [identify_shift_register_scan_segments](#) on page 710
- [identify_test_mode_registers](#) on page 712
- [insert_dft](#) on page 715
- [insert_dft_boundary_scan](#) on page 717
- [insert_dft_compression_logic](#) on page 721

Command Reference for Encounter RTL Compiler

Design for Test

- [insert_dft_dfa_test_points](#) on page 732
- [insert_dft_jtag_macro](#) on page 736
- [insert_dft_lockup_element](#) on page 740
- [insert_dft_logic_bist](#) on page 741
- [insert_dft_mbist](#) on page 743
- [insert_dft_opcg](#) on page 749
- [insert_dft_ptam](#) on page 751
- [insert_dft_rrfa_test_points](#) on page 754
- [insert_dft_scan_power_gating](#) on page 760
- [insert_dft_shadow_logic](#) on page 763
- [insert_dft_test_point](#) on page 768
- [insert_dft_user_test_point](#) on page 773
- [insert_dft_wrapper_cell](#) on page 775
- [insert_dft_wrapper_mode_decode_block](#) on page 780
- [map_mbist_cgc_to_cgic](#) on page 782
- [read_dft_abstract_model](#) on page 783
- [read_io_speclist](#) on page 785
- [replace_opcg_scan](#) on page 786
- [replace_scan](#) on page 787
- [report_dft_chains](#) on page 788
- [report_dft_core_wrapper](#) on page 789
- [report_dft_registers](#) on page 790
- [report_dft_setup](#) on page 791
- [report_dft_violations](#) on page 792
- [report_scan_compressibility](#) on page 793
- [report_test_power](#) on page 794
- [reset_opcg_equivalent](#) on page 795

Command Reference for Encounter RTL Compiler

Design for Test

- [reset_scan_equivalent](#) on page 796
- [set_compatible_test_clocks](#) on page 797
- [set_opcg_equivalent](#) on page 799
- [set_scan_equivalent](#) on page 801
- [update_scan_chains](#) on page 803
- [write_atpg](#) on page 805
- [write_bsdl](#) on page 808
- [write_compression_macro](#) on page 811
- [write_dft_abstract_model](#) on page 818
- [write_dft_rtl_model](#) on page 821
- [write_et_atpg](#) on page 822
- [write_et_bsv](#) on page 830
- [write_et_dfa](#) on page 833
- [write_et_lbist](#) on page 837
- [write_et_mbist](#) on page 840
- [write_et_rrfa](#) on page 844
- [write_io_speclist](#) on page 848
- [write_mbist_testbench](#) on page 852
- [write_scandef](#) on page 856

add_opcg_hold_mux

```
add_opcg_hold_mux  
    -edge_mode test_signal  
    -instance instance  
    [design]
```

Replaces the specified domain blocking scan flop instance with its OPCG-equivalent if you specified OPCG-equivalency mappings with the `set_opcg_equivalent` command. If the OPCG-equivalency mappings are not available, or if the scan cell type of the instance does not have an entry in the OPCG-equivalency table, the command adds a hold mux before the scan cell to convert it to an OPCG-equivalent cell and makes all necessary connections.

Options and Arguments

design Specifies the design in which you want to replace an instance.

-edge_mode *test_signal*
Specifies the global edge-mode signal to connect.

-instance *instance* Specifies the instance to be replaced.

Related Information

Affected by these commands: [reset opcq equivalent](#) on page 795
[set opcq equivalent](#) on page 799

analyze_scan_compressibility

```
analyze_scan_compressibility
    -library string [design] [-chains integer]
    [-minimum_scanned_flop_percentage integer]
    [-compressor {xor | misr | mimic_bidi_misr}]
    [-decompressor {broadcast | xor}]
    [-mask {wide1 | wide0 | wide2}] [-serial_misr]
    [-effort {low | medium | high}]
    [-fault_sample_size fault_sample_size]
    [-ratio_list list_of_compression_ratios]
    [-directory directory] [-dont_run_atpg]
    [-atpg_options string] [-build_model_options string]
    [-build_faultmodel_options string]
    [-build_testmode_options string]
    [-verify_test_structures_options string] [-verbose]
```

Analyzes a design for scan-based compressibility and produces actual compression results for each compression setting. Analysis is done based on the scan chain information in the design.

1. If actual scan chains exist, analysis is performed based on the number of actual scan chains.
2. If actual scan chains do not exist, analysis is performed based on the number of user-defined scan chains.
3. If user-defined scan chains do not exist, analysis is performed based on the number of chains specified using the `-chains` option.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-atpg_options string`

Specifies a string containing the extra options to run ATPG-based testability analysis.

Use the following format for the *string*:

`"option1=value option2=value ..."`

Note: For more information on these options, refer to the `create_tests` command in the *Command Line Reference* (of the Encounter Test documentation).

Command Reference for Encounter RTL Compiler

Design for Test

`-build_faultmodel_options string`

Specifies a string containing the extra options to build a fault model.

Note: For more information on these options, refer to the `build_faultmodel` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_model_options string`

Specifies extra options to apply when building the Encounter Test model.

Use the following format for the *string*:

`"option1=value option2=value ..."`

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_testmode_options string`

Specifies extra options to apply when building the test mode for Encounter Test.

Use the following format for the *string*:

`"option1=value option2=value ..."`

Note: For more information on these options, refer to the `build_testmode` command in the *Command Line Reference* (of the Encounter Test documentation).

`-chains integer`

Specifies the number of scan chains to be analyzed. This option is only required if no actual or user-defined scan chains exist.

The specified number must be equal to or larger than one.

`-compressor {xor | misr | mimic_bidi_misr}`

Specifies the type of compression logic for analysis:

- `xor` analyzes an XOR-based compressor
- `misr` analyzes a MISR-based compressor
- `mimic_bidi_misr` analyzes an on-product MISR compressor and mimics the behavior of the design as though the pads can be configured bidirectionally.

Default: xor

Command Reference for Encounter RTL Compiler

Design for Test

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic for analysis:

- `broadcast` specifies broadcast-based decompression logic (simple scan fanout).
- `xor` specifies an XOR-based spreader network in addition to the broadcast-based decompression logic.

Default: broadcast

design

Specifies the name of the top-level design on which to perform analysis.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory atpg_directory`

Specifies the directory to run and store ATPG results.

Note: ATPG results tend to consume high amounts of disk space for larger designs.

Default: `current_working_directory/asc`

`-dont_run_atpg`

Specifies to only insert the different compression options and produces a list of the ATPG jobs to be run.

Note: This option can be used in combination with the `report_scan_compressibility` command as the list of ATPG jobs is produced. The ATPG jobs may be run with either of the following methods:

- Run on a different workstation.
- Run in parallel mode to reduce runtime if a Load Sharing Facility (LSF) environment is available.



Neither of the preceding methods are automatically invoked by the `analyze_scan_compressibility` command or the `report_scan_compressibility` command.

`-effort {low | medium | high}`

Specifies the effort to be used for ATPG.

Command Reference for Encounter RTL Compiler

Design for Test

Note: Higher effort levels on larger designs cause increased runtimes.

Default: low

`-fault_sample_size integer_sample_size`

Specifies the number of faults to simulate to predict test coverage. This number affects the number of partitions or slices to be run depending on the total number of faults in the design. Specifying a higher number obtains more accurate results while specifying a lower number reduced runtime.

Defaults: 20000. For full ATPG, -1.

`-library string`

Specifies the list of Verilog structural library files.

You can specify the files explicitly or you can specify an *include* file that lists the files. You can also specify directories of Verilog files but you cannot reference directories in an include file.

For example, assume the following Verilog files are required:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

You can specify the files in either of the following ways:

1. Explicitly:

```
analyze_scan_compressibility -library "./padcells.v \  
./stdcells.v ./memories ./ip_blocks" ...
```

2. Using an include file.

```
analyze_scan_compressibility \  
-library "include_libraries.v ./memories \  
./ip_blocks" ...
```

where you created a file `include_libraries.v` with the following contents:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

Command Reference for Encounter RTL Compiler

Design for Test

`-mask {wide1 | wide0 | wide2}`

Specifies the scan channel masking logic type for analysis. The masking types that can be used depend on the compressor type specified with the `-compressor` option.

By default, no masking logic is inserted.

RTL Compiler provides three types of channel masking: `wide0`, `wide1`, and `wide2`.

- For XOR-based compression, two types of masking are supported: `wide1` and `wide2`
- For MISR-based compression, three type of masking are supported: `wide0`, `wide1`, and `wide2`.

When requested to insert `wide1` masking, one mask register bit is added per channel. When requested to insert `wide2` masking, two mask register bits are added per channel.

`-minimum_scanned_flop_percentage integer`

Specifies the minimum percentage of scannable flops required to allow the scan compression analysis to proceed.

Default: 95

`-ratio_list list_of_compression_levels`

Specifies the list of compression ratios to be analyzed.

Default: "20 50 100".

`-serial_misr_read` Performs analysis for serial MISR compression.

You can only specify this option for a MISR-based compression (`-compressor` option is set to `misr`).

You cannot specify this option when the `-decompressor` option is set to `xor`.

`-verbose` Displays all messages during analysis of the design for scan-based compressibility.

`-verify_test_structures_options string`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Use the following format for the `string`:

`"option1=value option2=value ..."`

Command Reference for Encounter RTL Compiler

Design for Test

Note: For more information on these options, refer to the `verify_test_structures` command in the *Command Line Reference* (of the Encounter Test documentation).

Examples

- The following command performs XOR-based compression analysis with `widel` masking for a ratio list of "20 50 100". The command does not perform the ATPG runs of the different compression schemes because the `-dont_run_atpg` option was specified.

```
rc:/> analyze_scan_compressibility -chains 8 -compressor xor -mask widel \
    -ratio_list "20 50 100" -dont_run_atpg -library mylibrary -directory asc
Starting scan compressibility analysis for module 'test'

Creating work directory asc ......

Analyzing the design with the following configuration
-----
Design      - test
Decompressor - broadcast
Compressor   - xor
mask        - widel
ratio       - 20 50 100

Scan Registers Status
-----
(Number of Scannable Registers) / (Total Register Count) =
--> 999 / 999 = 100%

(Number of flops mapped to scan) = 999
(Number of flops passing tdrc)   = 999

(Number of Scannable Registers)
(Number of flops scan mapped and passing tdrc) +
(Number of flops in shift-register segments) +
(Number of flops in abstract scan segments) =
--> 999 + 0 + 0 = 999

(Total Register Count)
(Number of registers) +
(Number of flops in abstract scan segments) =
--> 999 + 0 = 999

(*) The percentage of scannable registers is 100%.

Building the Fullscan Chains
-----
Building balanced scan chains assuming all test clocks are compatible
and mixing of clock edges in scan chains is allowed.
If you prefer a different set of constraints, please build the scan
chains before calling analyze_scan_compressibility.

Configuring 8 chains for 999 scan f/f
...chain01 of length: 125
...chain02 of length: 125
...chain03 of length: 125
...chain04 of length: 125
...chain05 of length: 125
...chain06 of length: 125
...chain07 of length: 125
...chain08 of length: 124

Encounter Test scripts for ATPG have been written
for fullscan mode to directory 'asc/et_scripts_fullscan'.
```

Command Reference for Encounter RTL Compiler

Design for Test

```
.... Invoke the script from OS command line as 'et -e  
asc/et_scripts_fullscan/runet.atpg'.
```

Inserting Scan Compression Logic

```
-----  
  
Compressing scan chains with ratio 20  
Tool achieved ratio: 17.8  
  
Encounter Test scripts for ATPG have been written  
for compression mode with fullscan top-off to directory  
'asc/et_scripts_ratio_20'.  
.... Invoke the script from OS command line as 'et -e  
asc/et_scripts_ratio_20/runet.atpg'.  
  
-----  
  
Compressing scan chains with ratio 50  
Tool achieved ratio: 41.6  
  
Encounter Test scripts for ATPG have been written  
for compression mode with fullscan top-off to directory  
'asc/et_scripts_ratio_50'.  
.... Invoke the script from OS command line as 'et -e  
asc/et_scripts_ratio_50/runet.atpg'.  
  
-----  
  
Compressing scan chains with ratio 100  
Tool achieved ratio: 62.5  
  
Encounter Test scripts for ATPG have been written  
for compression mode with fullscan top-off to directory  
'asc/et_scripts_ratio_100'.  
.... Invoke the script from OS command line as 'et -e  
asc/et_scripts_ratio_100/runet.atpg'.  
  
Skipping ATPG because option '-dont_run_atpg' was specified Completed scan compressibility analysis for module 'test'
```

- The following command performs an XOR-based compression analysis, with wide1 masking, with an broadcast decompressor, and a ratio list of "10 20 30".

```
rc:> analyze_scan_compressibility -chains 1 \  
-library rules/tsmc13.v -ratio_list "10 20 30" -directory $outdir/newdirnr2  
Starting scan compressibility analysis for module 'DLX_CORE'  
  
using existing work directory: './newdirnr2'  
Encounter(TM) Test product version information:  
Encounter(R) Test and Diagnostics xxxx  
  
Analyzing the design with the following configuration  
-----  
Design      - DLX_CORE  
Decompressor - broadcast  
Compressor   - xor  
mask        - wide1  
ratio       - 10 20 30  
  
Scan Registers Status  
-----  
Replacing scan  
Scan flip-flops mapped for DFT          1348      100.00%  
(Number of Scannable Registers) / (Total Register Count) =  
--> 1348 / 1348 = 100%  
  
(Number of flops mapped to scan) = 1348  
(Number of flops passing tdrc) = 1348  
  
(Number of Scannable Registers)  
(Number of flops scan mapped and passing tdrc) +  
(Number of flops in shift-register segments) +  
(Number of flops in abstract scan segments) =  
--> 1348 + 0 + 0 = 1348  
  
(Total Register Count)  
(Number of registers) +  
(Number of flops in abstract scan segments) =  
--> 1348 + 0 = 1348
```

Command Reference for Encounter RTL Compiler

Design for Test

(*) The percentage of scannable registers is 100%.

Building the Fullscan Chains

Building balanced scan chains assuming all test clocks are compatible and mixing of clock edges in scan chains is allowed. If you prefer a different set of constraints, please build the scan chains before calling analyze_scan_compressibility.

Configuring 1 chains for 1348 scan f/f
...chain01 of length: 1348

Encounter Test scripts for ATPG have been written
for fullscan mode to directory './newdirnr2/et_scripts_fullscan'.

Inserting Scan Compression Logic

Compressing scan chains with ratio 10
Tool achieved ratio: 9.9

Encounter Test scripts for ATPG have been written
for compression mode with fullscan top-off to directory
'./newdirnr2/et_scripts_ratio_10'.

Compressing scan chains with ratio 20
Tool achieved ratio: 19.8

Encounter Test scripts for ATPG have been written
for compression mode with fullscan top-off to directory './newdirnr2/et_scripts_ratio_20'.

Compressing scan chains with ratio 30
Tool achieved ratio: 29.9

Encounter Test scripts for ATPG have been written
for compression mode with fullscan top-off to directory './newdirnr2/et_scripts_ratio_30'.

Running Encounter Test ATPG

Running ./newdirnr2/et_scripts_fullscan/runet.atpg
Running ./newdirnr2/et_scripts_ratio_10/runet.atpg
Running ./newdirnr2/et_scripts_ratio_20/runet.atpg
Running ./newdirnr2/et_scripts_ratio_30/runet.atpg

Results of analyze_scan_compressibility

Design - DLX_CORE
Decompressor - broadcast
Compressor - xor
mask - wide1

Achieved compression table with fullscan top-off vectors

IC	TATR	TDVR	ATCov.	CL	PAT-comp	PAT-fs	Cycles	Runtime	Gatecount	Area
fs	1.0	1.0	99.99%	1348	-	305	414762	00:00:03	-	-
10	5.6	5.6	99.72%	135	301	22	73532	00:00:04	55	680.6569999999992
20	5.5	5.5	99.45%	68	324	37	75711	00:00:05	112	1357.9199999999983
30	4.4	4.4	97.83%	45	299	57	95084	00:00:13	155	1941.8260000000001

Achieved compression table without fullscan top-off vectors

IC	TATR	TDVR	ATCov.	CL	PAT-comp	PAT-fs	Cycles
fs	1.0	1.0	99.99%	1348	-	305	414762
10	9.8	9.8	99.72%	135	301	-	42212
20	17.6	17.6	99.45%	68	324	-	23586
30	27.8	27.8	97.83%	45	299	-	14946

Total ATPG runtime: 00:00:25 (hrs:min:sec).

IC - Inserted compression
TATR - Test application time reduction
TDVR - Test data volume reduction
ATCov. - ATPG test mode coverage
CL - Channel Length
PAT-comp - Number of compression test patterns
PAT-fs - Number of fullscan test patterns
Runtime - ATPG runtime
fs - Uncompressed (Fullscan) run

Completed scan compressibility analysis for module 'DLX_CORE'

Command Reference for Encounter RTL Compiler

Design for Test

- The following command performs MISR-based compression analysis, with `widel` masking, a ratio list of 2, sixteen scan chains.

```
rc:> analyze_scan_compressibility -library mylibrary -chains 16 \
    -compressor misr -ratio_list 2 -mask widel -fault_sample_size 5000 -dir asco
```

Related Information

[Analyzing and Reporting Scan Compressibility in Design for Test in Encounter RTL Compiler](#).

Affected by these commands: [connect scan chains](#) on page 620

[define_dft_scan_chain](#) on page 674

Related command: [report scan_compressibility](#) on page 481

analyze_testability

```
analyze_testability [-library string]
    [-effort {low|medium|high}]
    [-atpg_options string]
    [-build_faultmodel_options string]
    [-build_model_options string]
    [-build_testmode_options string]
    [-fault_sample_size integer]
    [-etlog file] -directory string [design]
```

Invokes Encounter Test to perform Automatic Test Pattern Generator (ATPG) based testability analysis in either assume or fullscan mode.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-atpg_options *string*

Specifies a string containing the extra options to run ATPG-based testability analysis.

Note: For more information on these options, refer to the `create_tests` command in the *Command Line Reference* (of the Encounter Test documentation).

-build_faultmodel_options *string*

Specifies a string containing the extra options to build a fault model.

Note: For more information on these options, refer to the `build_faultmodel` command in the *Command Line Reference* (of the Encounter Test documentation).

-build_model_options *string*

Specifies a string containing the extra options to build a model.

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

-build_testmode_options *string*

Command Reference for Encounter RTL Compiler

Design for Test

Specifies a string containing the extra options to build the test modes which define the scan structure and active logic for testing.

Note: For more information on these options, refer to the `build_testmode` command in the *Command Line Reference* (of the Encounter Test documentation).

design Specifies the name of the top-level design on which you want to perform test analysis and test-point selection.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-directory *string*

Specifies the working directory for Encounter Test.

-effort {low | medium | high}

Specifies the effort to be used for the ATPG-based testability analysis.

Default: low

-et log file

Specifies the name of the Encounter Test log file. This file will be generated in the specified directory.

Default: eta from rc.log

-fault sample size *integer*

Specifies the approximate number of faults simulated to predict the test coverage. This number affects the number of partitions or slices to be run depending on the total number of faults in the design.

The default sample size gives a good estimation of fault coverage while limiting the run time. Use a higher number to get a better accuracy, or a smaller number to reduce your run time.

Default: 20 (K)

-library *string*

Specifies the list of Verilog structural library files, for example, `file1 file2`. Refer to the [`write_et_atpg -library`](#) option description for additional information.

Note: This option is only required when you invoke this command on a mapped netlist.

Examples

- The following example performs only ATPG-based testability analysis. The command generates a report on the fault coverage in the log file.

```
analyze_testability
```

- The following command instructs to build a model using the IEEE standard Verilog parser.

```
analyze_testability -build_mode_options "vlogparser=IEEEstandard"
```

Related Information

[Using Encounter Test Software to Analyze Testability in Design for Test in Encounter RTL Compiler.](#)

Affected by these constraints: [define_dft_test_mode](#) on page 697

[define_dft_test_clock](#) on page 693

check_atpg_rules

```
check_atpg_rules [-library string] [-compression]
[-directory string] [design]
```

Generates a template script to run Encounter Test to verify if the design and its test structures are ATPG-ready. More specifically, the generated script allows you to check for

- Tristate drivers for contention
 - These conditions might cause manufacturing test problems
- Feedback loops
 - Failure to break combinational feedback loops might cause reduced test coverage.
- Clock signal races
 - Checks for any flip-flop with a potential race condition between its data and clock signal.
- Test clock control

This command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis
- `et.modedef`—A file describing the test mode when running ATPG in `assumed scan` mode
- `topmodulename.ASSUMED.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- `topmodulename.FULLSCAN.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- If the ATPG-based testability analysis is run in compression mode, the following pin-assignment files are generated in addition to the `topmodulename.FULLSCAN.pinassign` file. In this case, All three files include the compression test signals with their appropriate test functions to validate their specific test mode:
 - ❑ `topmodulename.COMPRESSION_DECOMP.pinassign`—A file generated *only* when inserting XOR-based decompression logic
 - ❑ `topmodulename.COMPRESSION.pinassign`—A file generated to verify the broadcast-based decompression logic

Command Reference for Encounter RTL Compiler

Design for Test

- `runet.tsv`—A template script file for Encounter Test to verify the design and its test structures

Options and Arguments

<code>-compression</code>	Performs additional checks if the design has compression logic.
<code>design</code>	Specifies the name of the top-level design for which you want to check if it ATPG-ready. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
<code>-directory string</code>	Specifies the working directory for Encounter Test. Note: If the script files already exist, rerunning the command will overwrite them. <i>Default:</i> <code>./check_atpg_rules</code>
<code>-library string</code>	Specifies the list of Verilog structural library files, for example, <code>file1 file2</code> . Refer to the <u>write_et_atpg_library</u> option description for additional information. Note: This option is only required when you invoke this command on a mapped netlist.

Example

```
rc:/> check_atpg_rules
```

Encounter Test scripts to check whether a design is ATPG ready have been written to directory `'./check_atpg_rules'`.

Invoke the scripts as '`et -e ./check_atpg_rules/runet.tsv`'

check_dft_pad_configuration

```
check_dft_pad_configuration [design] [> file]
```

Checks and reports the data direction control of the pad logic for the test I/O ports.

Options and Arguments

<i>design</i>	Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
<i>file</i>	Specifies the file to which to redirect the detailed output. If no file is specified, the output is written to standard out (<code>stdout</code>) and to the log file.

Related Information

[Checking the Pad Configuration in Design for Test in Encounter RTL Compiler.](#)

check_dft_rules

```
check_dft_rules [design] [-advanced]
    [-max_print_violations integer] [> file]
    [-max_print_registers integer]
    [-max_print_fanin integer]
    [-dft_configuration_mode dft_config_mode_name]
```

Evaluates the design for DFT-readiness. Flip-flops that pass the DFT rule checks are later mapped to scan flip-flops during synthesis and included in a scan chain during scan connection. Flip-flops that fail the DFT rule checks and flip-flops marked with either a `dft_dont_scan` attribute or a `preserve` attribute, or flip-flops instantiated in lower-level blocks marked with a `preserve` attribute, are not mapped to scan flip-flops and are excluded from the scan chains. The DFT rule checker also analyzes the libraries and reports on the valid scan cells.

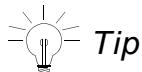
Table 10-1 lists the DFT rule violations that this command checks.

Table 10-1 Checking For and Auto-Fixing of DFT Rule Violations

DFT Rule Violation	Checked?	Auto-Fixed?
Uncontrollable asynchronous set or reset signals	Checked	Yes
Gated clocks and derived clocks	Checked	Yes
Flip-flop's clock port connected to tied lines	Checked	Yes
Conflicting clock and asynchronous set or reset signals	Checked	No
Tristate contention for internal and external nets	Checked(*)	Yes
Same asynchronous set or reset data race conditions	Checked(*)	Yes
Clock and data race conditions	Checked(*)	No
Floating nets violations	Checked(*)	No
X-source violations	Checked(*)	Yes
Floating conditions	Not checked	No

Note: (*) indicates that the check is performed using the `-advanced` option.

To maximize fault coverage, you should try to fix any DFT rule violations, so that all flip-flops can be included in a scan chain. You can either modify the RTL or use the DFT fix capabilities using the `fix_dft_violations` command.



To include the RTL file name and line number at which the DFT violation occurred in the messages produced by `check_dft_rules`, set the `hdl_track_filename_row_col root` attribute to `true` before elaboration.

You can find the objects created by the `check_dft_rules` command in:

`/designs/design/dft/test_clock_domains`

The detected violations are placed in:

`/designs/design/dft/report/violation`

Options and Arguments

`-advanced` Specifies to perform checking for tristate contention, same asynchronous set or reset data race conditions, clock and data race conditions, x-source generators, and floating nets violations.

`design` Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dft_configuration_mode dft_configuration_mode_name`

Specifies the object name of the scan mode to be checked. A scan mode is defined using the `define_dft_dft_configuration_mode` command.

Scan modes can be used to build the top-level scan chains with specific elements in different modes of operation (multi-mode), or when concatenating default scan chains into a single longer scan chain in a different scan mode of operation.

`file` Specifies the file to which to redirect the detailed output.

`-max_print_fanin integer`

Limits the number of pins and ports reported in the fanin cone for any violation.

Command Reference for Encounter RTL Compiler

Design for Test

Note: This option affects DFT checks performed for same asynchronous set or reset data race conditions, clock and data race conditions, and x-source generators.

Default: 20

`-max_print_registers integer`

Limits the number of registers reported for any violation.

Default: 20

`-max_print_violations integer`

Controls the number of DFT violations for which the details are printed to the screen and log file. Specify `-1` to write the details of all violations to the log file.

Default: 20

Examples

- The following example defines the shift-enable signal and its active polarity, then runs the DFT rule checker. The output of the `check_dft_rules` command is written to the `DFT.rules` file. The return value of the command (2) corresponds to the number of violations found.

```
rc:/> define_dft shift_enable scan_en -active high
rc:/> check_dft_rules > DFT.rules
    Checking DFT rules for 'top' module under 'muxed_scan' style

    Checking DFT rules for clock pins
    ...
    Checking DFT rules for async. pins
    ...

    Detected 2 DFT rule violation(s)
        ... see the log file for more details
    Number of user specified non-Scan registers:    0
        Number of registers that fail DFT rules:    4
        Number of registers that pass DFT rules:    1
    Percentage of total registers that are scannable: 20%
rc:/> sh more DFT.rules

    Checking DFT rules for 'top' module under 'muxed_scan' style
    Processing techlib techlib 25
        Identified a valid scan cell 'SDFFHQX1'
            active clock edge: rising
        Identified a valid scan cell 'SDFFHQX2'
            active clock edge: rising
        ...
        Identified 60 valid usable scan cells
    Detected 2 DFT rule violation(s)
        Summary of check_dft_rules
        ****
        Number of usable scan cells: 60
```

Command Reference for Encounter RTL Compiler

Design for Test

Clock Rule Violations:

```
-----  
Internally driven clock net: 1  
    Tied constant clock net: 0  
        Undriven clock net: 0  
Conflicting async/clock net: 0  
    Misc. clock net: 0
```

Async. set/reset Rule Violations:

```
-----  
Internally driven async net: 1  
    Tied active async net: 0  
        Undriven async net: 0  
Misc. async net: 0
```

Total number of DFT violations: 2
Clock Violation

0: internal or gated clock signal in module 'top', net 'Iclk', inst/pin 'g4/z' (file: test3.v, line 11) [CLOCK-05]

Effective fanin cone:

```
clk  
en
```

Async Violation

1: async signal driven by a sequential element in module 'top', net 'Iset', inst/pin 'Iset_reg/q' (file: test3.v, line 13) [ASYNC-05]

Effective fanin cone:

```
Iset_reg/q  
Violation # 0 affects 4 registers  
Violation # 1 affects 4 registers
```

Note - a register may be violating multiple DFT rules

Total number of Test Clock Domains: 1

DFT Test Clock Domain: clk

Test Clock 'clk' (Positive edge) has 1 registers

Number of user specified non-Scan registers: 0

Number of registers that fail DFT rules: 4

Number of registers that pass DFT rules: 1

Percentage of total registers that are scannable: 20%

- The following example shows tristate net checking with the `check_dft_rules -advanced` option.

```
rc:/> check_dft_rules -advanced  
Checking DFT rules for 'test' module under 'muxed_scan' style  
    Processing techlib tsmc_25 for muxed_scan scan cells  
    Identified 60 valid usable scan cells  
    Checking DFT rules for clock pins  
Info      : Added DFT object. [DFT-100]  
           : Added test clock domain 'clk'.  
Info      : Added DFT object. [DFT-100]  
           : Added test clock 'clk'.  
    Checking DFT rules for async. pins  
    Checking DFT rules for shift registers.  
    Checking DFT rules for tristate nets.  
    Checking DFT rules for clock data race conditions.  
    Checking DFT rules for set reset data race conditions.  
    Checking DFT rules for x-sources.  
Detected 1 DFT rule violation(s)  
    Summary of check_dft_rules  
*****  
Number of usable scan cells: 60
```

Command Reference for Encounter RTL Compiler

Design for Test

Clock Rule Violations:

```
-----  
Internally driven clock net: 0  
Tied constant clock net: 0  
Undriven clock net: 0  
Conflicting async & clock net: 0  
Misc. clock net: 0
```

Async. set/reset Rule Violations:

```
-----  
Internally driven async net: 0  
Tied active async net: 0  
Undriven async net: 0  
Misc. async net: 0
```

Advanced DFT Rule Violations:

```
-----  
Tristate net contention violation: 1  
Potential race condition violation: 0  
X-source violation: 0
```

Total number of DFT violations: 1

Warning : DFT Tristate net contention Violation. [DFT-315]
: # 0 <vid_0_tristate_net>: tristate net 'tbus' connected to pin
'out_reg/d' potentially driven by conflicting values [TRISTATE_NET-01]
: To remove the net contention violation in scan-shift mode, either modify the
RTL, or use the '-tristate_net' option of the 'fix_dft_violations' command.

Tristate net drivers:
i_block2/g1/z
i_block1/g1/z
i_block3/g1/z

Violations sorted by type and number of affected registers
Note - a register may be violating multiple DFT rules.

There are '1' tristate net violations.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Running the DFT Rule Checker](#)
- [Advanced DFT Rule Checking](#)
- [Defining Scan Configuration Modes](#)
- [Concatenating Scan Chains](#)

Affected by these constraints:

[define dft abstract segment](#) on page 628
[define dft dft configuration mode](#) on page 636
[define dft shift enable](#) on page 686
[define dft test clock](#) on page 693
[define dft test mode](#) on page 697

Affects these commands:

[connect scan chains](#) on page 620
[fix dft violations](#) on page 703
[report dft registers](#) on page 790
[synthesize](#) on page 348

Affected by these attributes:

[dft controllable](#)
[dft dont scan](#)
[dft identify test signals](#)
[dft identify top level test clocks](#)
[dft scan style](#)
(instance) [preserve](#)
(subdesign) [preserve](#)
[dft identify xsource violations from timing models](#)

Sets these attributes:

[dft status](#)
[dft violation](#)
[type](#)
[Violations Attributes](#)

check_mbist_rules

```
check_mbist_rules [-design design]
    [-dft_configuration_mode dft_config_mode_name]
    [-direct_access_only] [-interface_file_dirs string]
```

Checks for MBIST rule violations on the specified design.

Table 10-2 lists the MBIST rules checked by the command. The command does not auto-fix detected MBIST rules violations.

Table 10-2 Checked MBIST Rules

MBIST Rule

- The `Test_Control` test signal used during `insert dft_mbist` is properly controlled at the pin of all MBIST engines when the `dft_configuration_mode` is active.
 - All MBIST engine clock pins are properly controlled from design ports or internal pins for MBIST clocks defined with the `define_dft_mbist_clock -internal_clock_source` command
 - All MBIST interface files are consistent with the JTAG instructions and MBIST logic inserted in the design. Portions of the design which are blackboxes containing MBIST logic are limited to interface checks.
 - The need for a user supplied mode initialization sequence by downstream tools in case MBIST direct access is implemented using internal design pins as control signals.
-

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

- design *design* Specifies the name of the top-level design to be checked.
- dft_configuration_mode *dft_config_mode_name*
 - Specifies the configuration mode in which netlist tracing must be done.
 - Default: mbist*

Command Reference for Encounter RTL Compiler

Design for Test

-direct_access_only Specifies that the MBIST logic was inserted using the MBIST direct access control method, and no JTAG macro for MBIST operations.

Default: the design of the current directory in which the command is executed.

-interface_file_dirs dir1 dir2 ...

Checks the content of MBIST interface files against the MBIST logic inserted into the design.

Default: value of the `mbist_interface_files_location` attribute for the design specified with `-design`. If the attribute is not set, the directory defaults to `./mbist_design`.

Example

The following example shows how to define and use an MBIST configuration mode.

```
# Define the test signal states needed to establish the fullscan (ATPG) mode \
# of operation
define_dft test_mode -name TE      -active high TESTENABLE
define_dft test_mode -name TM0     -active high PORT00
define_dft test_mode -name TM1     -active low  PORT01
define_dft test_mode -name TM2     -active low  PORT02
define_dft test_mode -name TM3     -active high PORT03
define_dft test_mode -name TM4     -active low  PORT04

# Define the test signal states needed to establish the MBIST mode of operation
define_dft dft_configuration_mode -name My_MBIST_Mode \
    -mode_enable_high TE TM0 TM1 TM2 -mode_enable_low TM3 TM4

# Use the MBIST configuration mode to insert the MBIST logic and verify
# the MBIST rules

insert_dft mbist ... -directory ./My_MBIST_Dir \
    -dft_configuration_mode My_MBIST_Mode -test_control TM3

check_mbist_rules -dft_configuration_mode My_MBIST_Mode \
    -interface_file_dirs ./My_MBIST_Dir
```

Note: The logic states of the test signals specified in the MBIST configuration mode are decoded in the user-created logic and together with the test control signal of the `insert_dft_mbist` command establish the MBIST mode of operation.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Design Flows in “Inserting Memory Built-In-Self-Test Logic” in *Design for Test in Encounter RTL Compiler*](#)

Affected by these constraints: [define_dft_dft_configuration_mode](#) on page 636
[define_dft_mbist_clock](#) on page 656
[define_dft_mbist_direct_access](#) on page 659

Related attribute: [mbist_interface_files_location](#)

compress_block_level_chains

```
compress_block_level_chains
  {-ratio integer | -channel_length integer}
  [-chains actual_scan_chain]...
  [-decompressor {broadcast | xor}]
  [-compressor xor [-mask {wide1|wide2}]
   |-compressor misr [-mask {wide0 | wide1 | wide2}] ]
  [-mask_sharing_ratio integer]
  [-power_aware] [-target_period integer]
  [-preview] [-inside instance] [design]
```

Adds decompression and compression logic to reduce the effective length of the actual scan chains in the specified block. Use this command instead of the `compress_scan_chains` command when the block to be compressed will be modeled as a compressed block in the hierarchical compression flow.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-chains actual_scan_chain...`

Specifies the names of the actual scan chains to be compressed.

By default all actual scan chains are compressed.

Note: When inserting MISR logic, all scan chains must eventually be compressed.

`{-channel_length integer | -ratio integer}`

Controls the maximum length of the internal scan channels. You can either specify the maximum length directly or the tool can derive the length of the internal scan channels by dividing the longest actual scan chain length by the ratio.

Note: The `-channel_length` option is recommended for the hierarchical compression flow.

`-compressor {xor | misr}`

Specifies the type of compression logic to be built:

- `xor` specifies to build an XOR-based compressor

Command Reference for Encounter RTL Compiler

Design for Test

- misr specifies to build a MISR-based compressor

Default: xor

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- xor specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- broadcast specifies to build a broadcast-based decompression logic (simple scan fanout).

Default: broadcast

design

Specifies the name of the top-level design whose scan chains must be compressed.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-inside instance`

Specifies the instance in which to instantiate the compression logic.

By default, the compression logic is inserted as a hierarchical instance in the top-level of the design.

`-mask {wide0 | wide1 | wide2}`

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the `-compressor` option.

By default, no masking logic is inserted.

Note: The syntax indicates which types are available for each of the compressor types.

`-mask_sharing_ratio integer`

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the value specified for the compression ratio.

Note: This option is only valid with `wide1` and `wide2` masking.

`-power_aware`

Ensures that the mux logic added during compression obeys the boundaries of the power domains defined in the CPF file.

Command Reference for Encounter RTL Compiler

Design for Test

-preview Reports the requested ratio, the maximum original scan chain length, the maximum subchain length, and the number of internal scan channels that would be created without making modifications to the netlist. Use this option to verify your compression architecture prior to inserting the compression logic.

-target_period integer

Specifies a target clock period (in picoseconds) used to optimize the compression macro. If the value zero (0) is specified, synthesis is performed with a low effort compile, and without applying external (input/output delay) constraints.

Default: value for `test_clock` period of the scan chains being compressed

Example

The following command requests an XOR-based compression with masking logic of type 1 and decompression logic of type XOR for design core.

```
compress_block_level_chains -channel_length 7 -decompressor xor -mask wide1 core
```

Related Information

[Hierarchical Compression Flow in Design for Test in Encounter RTL Compiler](#)

Affected by these constraints: [define_dft_jtag_instruction](#) on page 645

[define_dft_shift_enable](#) on page 686

[define_dft_test_mode](#) on page 697

Affected by these commands: [connect_scan_chains](#) on page 620

Affects these commands: [report_dft_chains](#) on page 788

[write_dft_abstract_model](#) on page 818

Sets these attributes: [compressed](#)

[dft_compression_signal](#)

[dft_mask_clock](#)

[dft_misr_clock](#)

[type](#)

Command Reference for Encounter RTL Compiler

Design for Test

compress_scan_chains

```
compress_scan_chains {-ratio integer | -channel_length integer}
    [-chains actual_scan_chain]...
    [-auto_create] [-power_aware]
    [-decompressor {broadcast
        | xor [-spread_enable test_signal]}]
    [-compression_enable test_signal] [-target_period integer]
    [-master_control test_signal]
    [-compressor xor
        -mask {wide1|wide2} [-mask_clock {port|pin} [-allow_shared_clock]]
        [-mask_load test_signal] [-mask_enable test_signal]
        [-create_mask_or_misr_chain
            -mask_or_misr_sdi {pin|port}
            -mask_or_misr_sdo {pin|port} [-shared_output]]
        [-mask_sharing_ratio integer]
        [-apply_timing_constraints [-timing_mode_names mode_list]]
        [-write_timing_constraints file]
        [-low_pin_compression
            [-lpc_control shift_enable] [-shift_enable shift_enable]]]
    | -compressor misr
        [-serial_misr_read [-misr_observe test_signal]]
        [-misr_clock {port|pin}]
        [-misr_reset_enable test_signal | -misr_reset_clock test_signal]
        [-misr_read test_signal | -use_all_scan_ios_unidirectionally]
        [-misr_shift_enable test_signal]
        [-mask {wide0 | wide1 | wide2}]
        [-mask_sharing_ratio integer]
        [-mask_clock {port|pin} [-allow_shared_clock]] [-mask_load test_signal]
        [-mask_enable test_signal_list]
        [-create_mask_or_misr_chain
            -mask_or_misr_sdi {pin|port}
            -mask_or_misr_sdo {pin|port} [-shared_output]]
    | -compressor hybrid
        [-misr_bypass test_signal]
        [-serial_misr_read] [-misr_observe test_signal]
        [-misr_clock {port|pin}]
        [-misr_reset_enable test_signal | -misr_reset_clock test_signal]
        [-misr_shift_enable test_signal]
        [-mask {wide0 | wide1 | wide2}]
        [-mask_sharing_ratio integer]
        [-mask_clock {port|pin}] [-mask_load test_signal]
        [-mask_enable test_signal_list]
        [-create_mask_or_misr_chain
            -mask_or_misr_sdi {pin|port}
            -mask_or_misr_sdo {pin|port} [-shared_output]]]
    | -compressor smartscan_xor
        -smartscan_ratio integer
        [-gate_shared_compression_clock]
        [-smartscan_no_update_stage] [-smartscan_serial_only]
        [-smartscan_parallel_access test_signal]
```

Command Reference for Encounter RTL Compiler

Design for Test

```
[-smartscan_enable test_signal]
[-smartscan_pulse_width_multiplier {1|2|4}]
[-mask {wide1|wide2}] [-mask_clock {port|pin}]
[-mask_load test_signal]
[-mask_sharing_ratio integer]
[-apply_timing_constraints [-timing_mode_names mode_list]]
[-write_timing_constraints file]
[-preview] [-inside instance] [design]
[-jtag_control_instruction jtag_instruction]
    [-allow_multiple_jtag_control]
[-share_mask_enable_with_scan_in]
```

Adds decompression and compression logic to reduce the effective length of the actual scan chains.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-allow_multiple_jtag_control

When controlling the compression testmode from a JTAG macro, (that is, the `-jtag_control_instruction` option is specified), `compress_scan_chains` checks for the presence of other compression macros that are also JTAG controlled. RC-DFT does not automatically support such a configuration and therefore insertion of a second JTAG controlled compression macro is disallowed by default. Specify this option to bypass this check and insert additional JTAG controlled compression macros. When this option is specified, the tool assumes you will manually perform any additional stitching needed and will appropriately modify any files generated for Encounter Test.

Note: You must also specify the `-jtag_control_instruction` option with this option.

-allow_shared_clock

Allows the mask or MISR clock to be shared with an existing full-scan test clock.

Note: The mask or MISR clock can only be shared with a test clock if you added gating logic which prevents the scan flops from pulsing during the channel mask load or MISR reset sequences. Since functional clocks are typically used for scanning, this requirement means that the functional clocks must be gated during test.

 *Important*

When specified with the `-auto_create` option, a `-mask_load` pin is automatically created. The `mask_load` pin must additionally be used to gate off the clock being shared. If this gating logic is not added, the mask loading or MISR reset procedure will corrupt the test data in the design.

`-apply_timing_constraints`

Applies timing constraints to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, no additional constraints are applied.

`-auto_create`

Automatically creates the necessary test pins as top-level ports.

If you omitted any of the following options

`-compression_enable`, `-spread_enable`, `-mask_clock`,
`-mask_load`, `-mask_enable`, `-mask_or_misr_sdi`,
`-mask_or_misr_sdo`, `-misr_clock`, `-misr_observe`,
`-misr_reset_enable`, `-misr_read`, `-misr_bypass`,
`-smartscan_enable`, and
`-smartscan_parallel_access`, the appropriate ports will be created and named using the following format:

prefixOption

For example, `prefixcompression_enable`, where `prefix` is the value of the `dft_prefix` root attribute.

Command Reference for Encounter RTL Compiler

Design for Test

`-chains actual_scan_chain...`

Specifies the names of the actual scan chains to be compressed.

By default all actual scan chains are compressed.

Note: When inserting MISR logic, all scan chains must eventually be compressed.

`{-channel_length integer | -ratio integer}`

Controls the maximum length of the internal scan channels. You can either specify the maximum length directly or the tool can derive the length of the internal scan channels by dividing the longest actual scan chain length by the ratio.

Note: The `-channel_length` option is recommended for the hierarchical compression flow.

`-compression_enable test_signal`

Specifies the name of the test signal that enables configuring the actual scan chains in compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. If you request to build the compression logic with a master control signal (`-master_control`), the input port driving the compression enable signal can be an existing functional pin (specified through the `-shared_in` option of `define_dft test_mode`). If you do not specify the `-master_control` option, you must define the compression enable signal without the `-shared_in` option.

`-compressor {xor | misr | hybrid | smartscan_xor}`

Specifies the type of compression logic to be built:

- `xor` specifies to build an XOR-based compressor
- `misr` specifies to build a MISR-based compressor
- `hybrid` specifies to build a MISR compression with MISR bypass capability. Bypassing the MISR allows you to perform compression using just the XOR compressor.
- `smartscan_xor` specifies to include smartscan logic in the compression macro.

Command Reference for Encounter RTL Compiler

Design for Test

Default: xor

`-create_mask_or_misr_chain`

Specifies to build a separate full-scan chain for the mask and MISR registers.

Note: To place the mask or MISR registers in a separate chain, an additional scan data input pin and scan data output pin are needed. You can either specify these pins using the `-mask_or_misr_sdi` and `-mask_or_misr_sdo` options, or make sure that the `-auto_create` option is specified.

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- xor specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- broadcast specifies to build a broadcast-based decompression logic (simple scan fanout).

Default: broadcast

`design`

Specifies the name of the top-level design whose scan chains must be compressed. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-gate_shared_compression_clock`

Specifies that the clock to the compression macro is shared with the test clock to the scan chains and that the tool must insert gating logic on the test clock path. The test clock will be gated using the TEST_CLOCK_ENABLE pin on the compression macro.

Note: The TEST_CLOCK_ENABLE pin will always exist on the smartscan compression macro and can be used for the clock gating. If this gating logic is not added, the scan chains will be disturbed during the mask load and/or smartscan register load operations.

Note: If you omit this option, you must manually insert the gating logic. The tool will print a message to warn you that this connection must be made but will not perform any subsequent checks to ensure the gating logic has been correctly added.

Command Reference for Encounter RTL Compiler

Design for Test

-inside *instance* Specifies the instance in which to instantiate the compression logic.

By default, the compression logic is inserted as a hierarchical instance in the top-level of the design.

-jtag_control_instruction *jtag_instruction*

Specifies which JTAG instruction will be used to target the compression macro's test data register.

-lpc_control *shift_enable_signal*

Specifies a shift-enable signal used to control low pin count compression.

You cannot specify the same shift-enable signal for both the **-lpc_control** and **-shift_enable** options.

If you omit this signal, the tool will use one of the default shift-enable signals.

If no shift-enable pin exists, the tool creates an **LPC_CONTROL** shift-enable pin if the **-auto_create** option is specified.

-low_pin_compression

Reduces the number of compression control pins required by using encoded control signals.

-mask {wide0 | wide1 | wide2}

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the **-compressor** option.

By default, no masking logic is inserted.

Note: The syntax indicates which types are available for each of the compressor types.

-mask_clock {pin|port}

Specifies the clock that controls the mask registers.

Note: The input port associated with this option can be an existing functional pin. This clock cannot be shared with an existing full-scan test clock pin unless you also specify the **-allow_shared_clocks** option.

Command Reference for Encounter RTL Compiler

Design for Test

`-mask_enable test_signal`

Specifies the name of the test signal that controls whether mask bits should be applied during the current scan cycle.

For wide2 masking, two mask enable signals must be specified.

This option cannot be specified when the `-compressor` option is set to `smartscan_xor`.

Note: If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-mask_load test_signal`

Specifies the name of the test signal that enables loading of the mask data into the mask data registers.

Note: If the `mask_clock` is dedicated (that is, is only used for mask register loading), this signal is not needed. If this signal is shared (is used to clock the MISR or other logic in the circuit), this signal is needed to gate non mask load clock pulses from corrupting the mask registers. If the `mask_clock` is shared with other logic, you can use this signal to protect the shared logic from corruption during the mask load sequence.

`-mask_or_misr_sdi {pin|port}`

Specifies the scan data input pin or port of the mask or MISR chain.

Note: The input port associated with this option can be an existing functional pin.

`-mask_or_misr_sdo {pin|port}`

Specifies the scan data output pin or port of the mask or MISR chain.

Note: If the output port associated with this option is an existing functional pin, you must specify the `-shared_out` option.

Command Reference for Encounter RTL Compiler

Design for Test

`-mask_sharing_ratio integer`

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the value specified for the compression ratio.

Note: This option is only valid with `wide1` and `wide2` masking.

`-master_control test_signal`

Specifies the master control signal that gates the compression enable signal used for compression.

Note: This test signal must be dedicated for test and must have been defined using the `define_dft test_mode` command.

`-misr_bypass test_signal`

Specifies the test signal used to bypass the MISR-based logic. This test signal is required in hybrid compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_clock {pin|port}`

Specifies the clock that controls the MISR registers.

Note: This input port cannot be shared with an existing full-scan test clock unless it is only used to accumulate the MISR signature or if the `-allow_shared_clocks` option is specified. The `-misr_clock` option is only used to accumulate the MISR signature if there are separate `-mask_clock` and `-misr_reset_clock` signals specified.

`-misr_observe test_signal`

Specifies the test signal used to select Serial MISR Read. This is required when the `-serial_misr_read` option is specified unless `-auto_create` or `-jtag_control_instruction` is also specified.

Note: You must also specify the `-serial_misr_read` option with this option.

Command Reference for Encounter RTL Compiler

Design for Test

`-misr_read test_signal`

Specifies the test signal to configure any bidirectional scan I/O pads for MISR compression.

Note: This option is mutually exclusive with the `-use_all_scan_ios_unidirectionally` option. Using scan I/O bidirectionally during MISR compression is only available with the `-compressor misr` option. When using the `-compressor hybrid` option, all scan I/O are used unidirectionally.

`-misr_reset_clock test_signal`

Specifies a separate dedicated test signal that is used to asynchronously reset the MISR.

Note: This option is mutually exclusive with the `-misr_reset_enable` option.

`-misr_reset_enable test_signal`

Specifies the test signal used to reset the MISR registers.

Notes:

- This option is mutually exclusive with the `-misr_reset_clock` option.
- If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_shift_enable test_signal`

Specifies the test signal used to enable MISR accumulation during scan shifting. When this signal is de-asserted, the contents of the MISR register will not change.

Note: If this option is omitted, the default shift-enable test signal for the design is used. If this option is specified, you must have defined this test signal using the `define_dft shift_enable` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft shift_enable` command).

Command Reference for Encounter RTL Compiler

Design for Test

-power_aware	Ensures that the mux logic added during compression obeys the boundaries of the power domains defined in the CPF file.
-preview	Reports the requested ratio, the maximum original scan chain length, the maximum subchain length, and the number of internal scan channels that would be created without making modifications to the netlist. Use this option to verify your compression architecture prior to inserting the compression logic.
-serial_misr_read	Specifies to include support for reading MISR bits serially through the scan data pins.
-share_mask_enable_with_scan_in	Allows to share the mask enable port with a scan data in port. When you specify this option you enable insertion of an asymmetrical compression macro.
-shared_output	Specifies that the scan data output port of the dedicated mask or MISR chain must be shared with a functional port. The tool creates the additional logic required to share the port.
-shift_enable <i>shift_enable</i>	Specifies the shift-enable signal to be used for low pin count compression. If you omit this option, the tool will use the default shift-enable signal.
-smartscan_enable <i>test_mode_signal</i>	Specifies the test mode signal that enables the smartscan mode. You can omit this option if you specified the -auto_create option. The tool will create the smartscan_enable port on the compression macro and connect it to the primary input for this test mode signal. When this signal is active it ensures that the smartscan mode is ON. Currently when this signal is inactive, the smartscan flops are part of the fullscan chains only. When inactive, and in the compression modes, the smartscan flops are <i>not</i> part of the compression channels.

Command Reference for Encounter RTL Compiler

Design for Test

`-smartscan_no_update_stage`

Prevents the insertion of update registers between the deserializer and the decompressor. In this case, lockup latches are inserted between the deserializer flops and the decompressor.

You cannot specify this option when you have set the `-smartscan_pulse_width_multiplier` option to either 2 or 4 .

By default, the tool inserts update registers between the deserializer and the decompressor.

`-smartscan_parallel_access test_mode_signal`

Specifies the test mode signal to use for parallel access to the smartscan flops. You can omit this option if you specified the `-auto_create` option. The tool will create the `smartscan_parallel_access` port on the compression macro and connect it to the primary input for this test mode signal.

`-smartscan_pulse_width_multiplier {1|2|4}`

Determines whether to add clock divider logic to widen the clock pulse going to the scan chains. You can specify the following values:

- 1—no logic added
- 2—increases the scan path through the SmartScan clock controller with 1 bit
- 4—increases the scan path through the SmartScan clock controller with 2 bits

Default: 1

`-smartscan_ratio integer`

Specifies the number of parallel scan data input pins that correspond to a single serial scan data input pin. The number of defined (fullscan) chains must be an integral multiple of the specified smartscan ratio.

Command Reference for Encounter RTL Compiler

Design for Test

`-smartscan_serial_only`

Specifies to only insert the smartscan serial-only interface. The number of deserializer (and serializer) registers will match the number of the defined chains. When building the model for Encounter Test (during the `build_model` step in the `write_et_atpg` scripts), the tool will create the pseudo pins for the parallel interface.

`-spread_enable test_signal`

Specifies the name of the test signal that enables applying the input test data to an XOR-based spreader network.

Use this option when `-decompressor` is set to `xor`.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-target_period integer`

Specifies a target clock period (in picoseconds) used to optimize the compression macro. If the value zero (0) is specified, synthesis is performed with a low effort compile, and without applying external (input/output delay) constraints.

Default: value for `test_clock` period of the scan chains being compressed

`-timing_mode_names mode_list`

Specifies the timing modes for which to generate the additional timing constraints that apply to the compression control signals.

The timing modes are taken into account when you specify the `-apply_timing_constraints` and `-write_timing_constraints` options.

Note: This applies only to multi-mode designs. Modes are created with the `create_mode` command.

`-use_all_scan_ios_unidirectionally`

Disables use of bidirectional scan I/O for a MISR-based compressor.

Note: This option is mutually exclusive with the `-misr_read` option.

`-write_timing_constraints file`

Specifies the file to which to write the timing constraints applied to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization. The timing constraints are not written if you specify the `-preview` option.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, constraints for all modes are written to the file.

Examples

- The following command requests XOR-based compression logic without masking, and requests creation of the necessary ports for the required test signals. In this case only a compression enable signal is required and thus one test port is created.

```
rc:/> compress_scan_chains -ratio 5 -compressor xor -auto_create
Will create a test port for 'compression_enable'
Checking out license 'Encounter_Test_Architect'... (1 seconds elapsed)
...
```

- The following command requests XOR-based compression logic with masking logic of type `widel`, and requests creation of the necessary ports for the required test signals. In this case three extra test signals are required for the masking logic and thus four test ports are created.

```
rc:/> compress_scan_chains -ratio 5 -compressor xor -mask widel -auto_create
Will create a test port for 'compression_enable'
Will create a test port for 'mask_load'
Will create a test port for 'mask_enable'
Will create a test port for 'mask_clock'
Checking out license 'Encounter_Test_Architect'... (1 seconds elapsed)
....
```

- The following command requests an XOR-based compression with masking logic of type `widel`, with decompression logic of type `xor`, and requests creation of the necessary ports for the required test signals. Compared to the second example, one extra test signal—the `spread_enable` signal—is required for the decompression logic, and thus five test ports are created.

```
rc:/> compress -ratio 5 -compressor xor -decompressor xor -mask widel \
-autocreate
Will create a test port for 'compression_enable'
Will create a test port for 'spread_enable'
Will create a test port for 'mask_load'
Will create a test port for 'mask_enable'
Will create a test port for 'mask_clock'
Checking out license 'Encounter_Test_Architect'... (2 seconds elapsed)...
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following command requests a MISR-based compression with masking logic of type wide1, and requests creation of the necessary ports for the required test signals. Compared to the two second example, one extra test signal—the misr_reset_enable signal—is required to reset the MISR registers, and thus five test ports are created.

```
rc:/> compress -ratio 5 -compressor misr -mask wide1 -auto_create
Will create a test port for 'compression_enable'
Will create a test port for 'misr_reset_enable'.
Will create a test port for 'misr_clock'.
Info - Defaulting the misr shift enable signal to the default shift_enable /
designs/test/dft/test_signals/SE
Will create a test port for 'mask_load'
Will create a test port for 'mask_enable'
Info - will share the 'misr_clock' and the 'mask_clock' ...
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Inserting Test Compression Logic](#)
- [Low Pin Count Compression Using Encoded Compression Signals](#)
- [Reducing Pin Count for Compression](#)
- [Using Asymmetrical Scan Compression](#)

Affected by these constraints: [define_dft_jtag_instruction](#) on page 645

[define_dft_shift_enable](#) on page 686

[define_dft_test_mode](#) on page 697

Affected by these commands: [connect_scan_chains](#) on page 620

Affects this command: [report_dft_chains](#) on page 788

Sets these attributes: [compressed](#)

[dft_compression_signal](#)

[dft_mask_clock](#)

[dft_misr_clock](#)

[type](#)

concat_scan_chains

```
concat_scan_chains
  -name string
  -chains actual_scan_chains
  [-dft_configuration_mode dft_config_mode_name]
  [-preview] [design]
```

Inserts muxing logic into the scan path of actual scan chains such that the scan chains are concatenated to become a single, longer scan chain in a specific test mode of operation.

Note: An actual scan chain may be specified only once per test mode, that is, multiple configurations of the same scan chain in the same test mode are disallowed.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-chains scan_chains

Specifies the names of the actual scan chains to be concatenated; where the scan chains are concatenated in the specified order.

design

Specifies the name of the top-level design for which to concatenate the scan chains. Specify this name if you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-dft_configuration_mode dft_config_mode_name

Specifies the object name of the scan mode used to concatenate the actual scan chains.

-name string

Specifies the name of the concatenated chain.

-preview

Reports how the actual scan chains would be concatenated, but does not perform the concatenation.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Concatenating Scan Chains](#)
- [Controlling Scan Configuration](#)

Affected by these constraints: [define_dft dft_configuration_mode](#) on page 636

[define_dft shift_enable](#) on page 686

[define_dft test_mode](#) on page 697

[set_compatible_test_clocks](#) on page 797

Affected by this command: [check_dft_rules](#) on page 589

Affects these commands: [report_dft_chains](#) on page 788

[write_dft_abstract_model](#) on page 818

[write_scandef](#) on page 856

Sets these attributes: [Actual Scan Chain](#) attributes

[Actual Scan Segment](#) attributes

Affected by these attributes: [decoded_pin](#)

[dft_lockup_element_type](#)

[dft_mix_clock_edges_in_scan_chains](#)

configure_pad_dft

```
configure_pad_dft -mode {input | output | tristate}  
    -test_control test_signal port
```

Inserts the required logic to configure the data direction control for a bidirectional or tristate pad during test mode.

Note: This command can configure a generic pad.

Options and Arguments

`-mode {input | output | tristate}`

Specifies in which mode the pad must be configured in test mode.

`input` Specifies to configure the pad in input mode.

`output` Specifies to configure the pad in output mode.

`tristate` Specifies to disable the pad.

`port` Specifies the top-level port that is connected to the I/O pad that the RC-DFT engine needs to configure.

`-test_control test_signal`

Specifies the test signal to use to control the pad.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Configuring Bidirectional and Tristate Pads in Test Mode](#)
- [Using a Functional Data Pin to Drive a Shift-Enable Test Signal](#)

Affected by these constraints: [define_dft shift_enable](#) on page 686

[define_dft test_mode](#) on page 697

Affects these commands: [check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

connect_compression_clocks

```
connect_compression_clocks  
  [-mask_clock test_clock] [-misr_clock test_clock]  
  [design]
```

Connects the compression clocks that were auto-created using the `compress_block_level_chains` command at the block level to the compression clocks at the top level. This command applies only to the hierarchical compression flow.

Options and Arguments

design Specifies the name of the design for which to connect the compression clocks.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

This option is required if multiple designs are loaded.

-mask_clock test_clock

Specifies the top-level clock that controls the mask registers.

If you omit this option, the tool uses the top-level test clock whose `dft_mask_clock test_clock` attribute is set to `true`.

-misr_clock test_clock

Specifies the top-level clock that controls the MISR registers.

If you omit this option, the tool uses the top-level test clock whose `dft_misr_clock test_clock` attribute is set to `true`.

Related Information

[Hierarchical Compression Flow in Design for Test in Encounter RTL Compiler](#)

Affected by these commands: [compress_block_level_chains](#) on page 598
 [compress_scan_chains](#) on page 601

connect_opcg_segments

```
connect_opcg_segments  
    [-chains actual_scan_chains]  
    [-preview] [design]
```

Connects the scan segments associated with the OPCG logic into the actual scan chains. You must run this command after you have connected the scan chains.

Options and Arguments

design Specifies the name of the design for which to connect the OPCG segments.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

This option is required if multiple designs are loaded.

-chains *actual scan chains*

Specifies the names of the actual scan chains to which the OPCG segments must be prepended.

-preview

Shows the potential connections for the OPCG segments, without making any modifications to the netlist.

Related Information

Connecting the OPCG Segments in Design for Test in Encounter RTL Compiler

Affected by these commands

[connect scan chains](#) on page 620

insert dft opca on page 749

connect_scan_chains

```
connect_scan_chains [design]
  [-preview] [-auto_create_chains]
  [-incremental] [-chains chain_list]
  [-elements element_list]
  [-keep_connected_SE]
  [-dont_exceed_min_number_of_scan_chains]
  [-pack | -create_empty_chains]
  [-dft_configuration_mode dft_config_mode_name]
  [-physical]
  [-power_domain power_domain_list] [-update_placement]
```

Configures and connects scan flip-flops which pass the DFT rule checks into scan chains. This command works at the current level of the hierarchy and all lower hierarchies instantiated in this module. The design must be mapped to the target library before connecting scan chains in a design.

The command returns the number of scan chains that the scan configuration engine creates (or would create if you use the `-preview` option).

You can find the objects created by the `connect_scan_chains` command in:

```
/designs/design/dft/actual_scan_chains
/designs/design/dft/actual_scan_segments
```

Options and Arguments

`-auto_create_chains` Allows the scan configuration engine to add new chains that are not defined through a `define_dft_scan_chain` constraint.

Without this option, the scan configuration engine reports an error if it needs more scan chains than have been defined with the `define_dft_scan_chain` command.

`-chains chain_list` Connects only the specified user-defined chain names. If the list is empty, none of the user-defined chains can be connected at this time. New chains are created if you specify the `-auto_create_chains` option.

The specified user-defined chains must have been defined using a `define_dft_scan_chain` constraint.

If omitted, all user-defined chains can be connected.

Command Reference for Encounter RTL Compiler

Design for Test

`-create_empty_chains`

Allows the scan configuration engine to create empty scan chains by making a direct connection from their scan data input to their scan data output if the number of scan chains to be configured is less than the minimum number of scan chains required in the design.

Note: Do not use this option when configuring scan chains to be used as internal scan channels that are loaded and unloaded using on-chip compression logic.

If this option is not specified, the scan configuration engine will move scan flops between compatible chains to satisfy the minimum number of scan chains requirement. This can result in configured scan chains having a sequential depth of one element.

`-dft_configuration_mode dft_configuration_mode_name`

Specifies the scan mode for which to build the scan chains.

`-dont_exceed_min_number_of_scan_chains`

Specifies to use the exact same number of scan chains as specified by the `dft_min_number_of_scan_chains` attribute when building the scan chain.

design

Specifies the name of the top-level design to be checked. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-elements element_list`

Considers only the specified elements for scan chain connection. An element can be a flip-flop, segment, or a hierarchical instance.

If you specify a hierarchical instance, all flops in this hierarchical instance that pass the DFT rule checker and that are mapped to scan for DFT, will be added to the chains.

Command Reference for Encounter RTL Compiler

Design for Test

If some of the scan flops in a hierarchical instance belong to a segment that crosses the boundary of this instance, these scan flops will only be connected if the remaining elements of the segment are also specified with the `-elements` option—either directly or indirectly through another hierarchical instance.

Note: If you specify this option with the `-power_domain` option, the specified elements must belong to the specified power domains.

`-incremental`
Adds new chains in incremental mode, without changing already connected scan chains stored in
`/designs/design/dft/report/actual_scan_chains`
Do not use user-defined chains with the same names as the `actual_scan_chains`.

`-keep_connected_SE`
Ensures that a flop that is already connected to a shift-enable signal keeps this connection when connected in a scan chain.
If you omit this option, the tool can break the existing connection to the shift enable and connect the shift-enable pin of the flop to either the default shift-enable signal or to the shift-enable signal specified for the chain.

`-pack`
Packs the scan chains to their maximum limit instead of balancing the chains (that is, attempting to create chains with similar lengths).
You can specify a chain-specific constraint using the `-max_length` option of the `define_dft_scan_chain` command or a global constraint by setting the value of the `dft_max_length_of_scan_chains` attribute.

`-physical`
Specifies to use the placement locations of the scan flops to connect the scan chains.
The placement information is obtained from the DEF file read in with the `read_def` command.

`-power_domain power_domain_list`
Considers only the scan flops that belong to the specified power domain(s) for scan chain connection.

Command Reference for Encounter RTL Compiler

Design for Test

-preview	Reports how the scan chains will be connected, but makes no modifications to the netlist. Use this option to verify your scan-chain architecture prior to connecting the scan chains.
-update_placement	Specifies to update the placement of the DFT logic that is added during the scan chain connection process. Note: Use this option only in the physical flow after you have already placed the design using <code>synthesize -to_placed</code> . The option will be ignored if the design is not placed.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Controlling Scan Configuration](#)
- [Connecting the Scan Chains](#)
- [Library-Domain Aware Scan Chain Configuration](#)
- [Power-Domain Aware Scan Chain Configuration](#)
- [Physical Scan Chain Synthesis](#)
- [Defining Scan Configuration Modes](#)

Related commands:

[write_compression_macro](#) on page 811

Affected by these constraints:

[define_dft_abstract_segment](#) on page 628

[define_dft_dft_configuration_mode](#) on page 636

[define_dft_fixed_segment](#) on page 641

[define_dft_floating_segment](#) on page 643

[define_dft_preserved_segment](#) on page 671

[define_dft_scan_chain](#) on page 674

[define_dft_scan_clock_a](#) on page 680

[define_dft_scan_clock_b](#) on page 683

[define_dft_shift_enable](#) on page 686

[define_dft_shift_register_segment](#) on page 689

[set_compatible_test_clocks](#) on page 797

Command Reference for Encounter RTL Compiler

Design for Test

Affected by these commands: [check_dft_rules](#) on page 589

[fix_dft_violations](#) on page 703

[synthesize](#) on page 348

Affects this command: [concat_scan_chains](#) on page 615

[report_dft_chains](#) on page 788

Sets these attributes: [Actual_Scan_Chain](#) attributes

[Actual_Scan_Segment](#) attributes

Affected by these attributes: [decoded_pin](#)

[dft_lockup_element_type](#)

[dft_max_length_of_scan_chains](#)

[dft_min_number_of_scan_chains](#)

[dft_mix_clock_edges_in_scan_chains](#)

[dft_prefix](#)

[dft_scan_map_mode](#)

define_dft

```
define_dft {abstract_segment | fixed_segment  
| boundary_scan_segment | dft_configuration_mode  
| domain_macro_parameters | jtag_macro  
| floating_segment | mbist_clock ! mbist_direct_access  
| opcg_domain | opcg_mode | opcg_trigger | osc_source  
| preserved_segment | scan_chain  
| scan_clock_a | scan_clock_b | shift_enable  
| shift_register_segment | tap_port | test_clock  
| test_mode }
```

Defines a DFT object. A DFT object can be a test signal, scan segment, or scan chain.

Options and Arguments

abstract_segment	Defines an abstract scan-chain segment object.
boundary_scan_segment	Defines a boundary-scan segment object.
dft_configuration_mode	Defines a scan mode for DFT configuration purposes.
domain_macro_parameters	Defines a set of domain macro parameters used to configure and build the OPCG domain macro logic.
fixed_segment	Defines a fixed scan-chain segment object.
floating_segment	Defines a floating scan-chain segment object.
jtag_macro	Defines a pre-instantiated third-party JTAG Macro.
jtag_instruction	Defines a user-defined instruction that is serially loaded into a boundary scan device.
jtag_instruction_register	Customizes the instruction register to allow adding user-defined instructions.
mbist_clock	Defines an MBIST clock object.
mbist_direct_access	Defines MBIST direct access interface pins or ports.
opcg_domain	Defines an OPCG (clock) domain.
opcg_mode	Defines an OPCG mode for Encounter Test ATPG.
opcg_trigger	Defines the trigger signal that will enable the trigger macro associated with the specified oscillator source.

Command Reference for Encounter RTL Compiler

Design for Test

osc_source	Defines an oscillator source on the output pin of a PLL instance that will drive the OPCG logic.
preserved_segment	Defines a preserved scan-chain segment object.
scan_chain	Defines a scan chain.
scan_clock_a	Defines the scan clock of the master latch for the LSSD scan style.
scan_clock_b	Defines the scan clock of the slave latch for the LSSD scan style.
shift_enable	Defines a <code>test_signal</code> object of type <code>shift_enable</code> .
shift_register_segment	Defines a shift register scan-chain segment object.
tap_port	Defines a TAP port (JTAG signal).
test_clock	Defines a test clock object.
test_mode	Defines a <code>test_signal</code> object of type <code>test_mode</code>

Related Information

Related commands:

- [define_dft_abstract_segment](#) on page 628
- [define_dft_boundary_scan_segment](#) on page 633
- [define_dft_dft_configuration_mode](#) on page 636
- [define_dft_domain_macro_parameters](#) on page 639
- [define_dft_fixed_segment](#) on page 641
- [define_dft_floating_segment](#) on page 643
- [define_dft_jtag_instruction](#) on page 645
- [define_dft_jtag_instruction_register](#) on page 649
- [define_dft_jtag_macro](#) on page 651
- [define_dft_mbist_clock](#) on page 656
- [define_dft_mbist_direct_access](#) on page 659
- [define_dft_opcg_domain](#) on page 662
- [define_dft_opcg_mode](#) on page 665
- [define_dft_opcg_trigger](#) on page 667

Command Reference for Encounter RTL Compiler

Design for Test

[define dft osc source](#) on page 669
[define dft preserved segment](#) on page 671
[define dft scan chain](#) on page 674
[define dft scan clock a](#) on page 680
[define dft scan clock b](#) on page 683
[define dft shift enable](#) on page 686
[define dft shift register segment](#) on page 689
[define dft tap port](#) on page 691
[define dft test clock](#) on page 693
[define dft test mode](#) on page 697

define_dft abstract_segment

```
define_dft abstract_segment [-name segment_name]
    {-module subdesign|-instance instance|-libcell cell}
    -sdi subport [-inversion] -sdo subport [-tail_inversion]
    -clock_port subport [-rise|-fall] [-off_state {high|low}]
    [-tail_clock_port subport
        [-tail_edge_rise | -tail_edge_fall]
        [-tail_clock_off_state {high|low}] ]
    [-other_clock_port subport
        [-other_clock_edge {rise|fall}]]...
    { { -shift_enable_port subport -active {high|low}
        | -connected_shift_enable }
    | { -scan_clock_a_port subport -scan_clock_b_port subport
        | -connected_scan_clock_a -connected_scan_clock_b } }
    [-test_mode_port subport
        -test_mode_active {low|high} ]...
    -length integer [-skew_safe]
    [-dft_configuration_mode dft_config_mode_name]
```

Defines an abstract segment. An abstract segment can be defined for objects of type blackbox, logic abstract module, or libcell timing model.

An abstract segment is a user-specified scan segment used at the next level of integration to define the sets of scan chains previously created for the object.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft abstract_segment` constraints in:

`/designs/top_design/dft/scan_segments`

Options and Arguments

`-active {low|high}` Specifies the active value for the shift-enable port.

`-clock_port subport`

Specifies the clock port—at the boundary of the blackbox or logic abstract module—driving the flip-flops at the head of the segment.

`-connected_scan_clock_a (-connected_scan_clock_b)`

Indicates that the `scan_clock_a` (`scan_clock_b`) port of the module boundary is driven by external logic (preconnected).

The external logic connected to the `scan_clock_a` (`scan_clock_b`) pin of the module will not be modified by the scan configuration engine.

Command Reference for Encounter RTL Compiler

Design for Test

Note: This option applies only for the clocked LSSD scan style.

`-connected_shift_enable`

Indicates that the shift enable port of the module boundary is driven by external logic (preconnected) or that the shift enable signal is internally generated within the module boundary. In either case, the external logic connected to the shift enable pin of the module, or the internal logic driving the shift enable pins of the flip-flops in the module will not be modified by the scan configuration engine.

This option cannot be specified together with the `-shift_enable` option.

`-dft_configuration_mode dft_configuration_mode_name`

Specifies in which scan mode the abstract segment will be connected in the scan chains.

`-instance instance` Specifies the instance name of the module for which the abstract segment is defined.

`-inversion` Indicates an inversion at the scan data input pin of the abstract segment.

`-length integer` Specifies the length of the abstract segment.

`-libcell cell` Specifies the library cell for which the abstract segment is defined. This option applies to library cells that are implemented as timing models and whose description includes the relevant test-related pins (such as scan data input and output, clock, shift-enable) to infer the scan chain architecture.

`-module subdesign` Specifies the subdesign (module) to which the element belongs.

`-name segment_name` Defines a name for the segment that you can use to reference in the [define_dft_scan_chain](#) constraint.

`-off_state {high|low}`

Specifies the off state of the system clock specified through the `-clock_port` option.

Note: This option applies only to the clocked LSSD scan style.

`-other_clock_edge {rise | fall}`

Specifies the active edge of the clock specified through the `-other_clock_port` option.

Command Reference for Encounter RTL Compiler

Design for Test

Default: rise

`-other_clock_port subport`

Specifies the clock port—at the boundary of the blackbox or logic abstract module—of another system clock used to drive some flip-flops in the abstract segment.

This option is only required if the clock is different from the clock specified through the `-clock_port` option or the `-tail_clock_port` option.

`{-rise | -fall}` Specifies the active edge of the clock specified through the `-clock_port` option.

Default: -rise

`-scan_clock_a_port (-scan_clock_b_port) subport`

Specifies the `scan_clock_a` (`scan_clock_b`) port at the boundary of the blackbox or logic abstract module to which the segment belongs. Specify this option to have the `connect_scan_chains` command make the connection from the top-level `scan_clock_a` (`scan_clock_b`) signals to the `scan_clock_a` (`scan_clock_b`) port of the module.

Note: This option applies only for the clocked LSSD scan style.

`-sdi (-sdo)` Specifies the scan data input (scan data output) of the segment.

- For a segment in a blackbox, specify a subport (port of the blackbox or logic abstract module).
- For a segment defined for a libcell, specify a pin of the libcell.

`-shift_enable_port subport`

Specifies the shift enable port at the boundary of the blackbox or logic abstract module to which the segment belongs. Specify this option if you want the `connect_scan_chains` command to make the connection from the top-level shift-enable signals to the shift-enable ports of the modules.

This option cannot be specified together with the `-connected_shift_enable` option.

`-skew_safe`

Indicates whether the abstract segment has a data lockup element connected at the end of its scan chain.

Command Reference for Encounter RTL Compiler

Design for Test

`-tail_clock_off_state {high|low}`

Specifies the off state of the system clock specified through the `-tail_clock_port` option.

Note: This option applies only to the clocked LSSD scan style.

`-tail_clock_port port`

Specifies the clock port—at the boundary of the blackbox or logic abstract module—driving the flip-flops at the tail of the segment. This option is only required if the clock used at the tail of the abstract segment is different from the clock specified through the `-clock_port` option.

`-tail_edge_rise | -tail_edge_fall`

Specifies the active edge of the clock specified through the `-tail_clock_port` option.

Default: `-tail_edge_rise`

`-tail_inversion`

Indicates an inversion at the scan data output pin of the abstract segment.

`-test_mode_active {low | high}`

Specifies the active value for the test-mode port.

This option should immediately follow the corresponding `-test_mode_port` option.

`-test_mode_port subport`

Specifies the test mode port at the boundary of the blackbox module to which the segment belongs.

Specify this option when the block-level design includes test-mode activated logic.

When the test-mode signals are specified, the propagated values of the top-level test-mode signals must match the expected block-level test-mode values and the segment must also pass the clock-controllability rule checks in order for the segment to be included into a top-level scan chain.

Note: The tool does not make connections to the test-mode ports of the block-level design. The connections should already exist in the netlist.

Examples

- The following example defines an abstract segment with length 3 in blackbox module b. The clock driving the flip-flops at the tail of the segment is the same as the clock driving the first elements in the segment.

```
rc:/> define_dft abstract_segment -name a1 -module b -length 3 \
-sdi {p4[0]} -sdo {p5[0]} -shift_enable {p6[0]} -active high -clock p3 -rise
```

- The following example defines an abstract segment in a timing model reference COMBELEM with length 20.

```
rc:/> define_dft abstract_segment -name combElem_seg -libcell COMBELEM \
-sdi A -sdo Z -shift_enable_port B -active hi -clock_port D2 -rise -length 20
```

- The following example defines an abstract segment ABS with length 10 in instance o1. The same clock clk with active rising edge is used for all flip-flops of the segment.

```
define_dft abstract_segment -instance o1 -sdi sdi -sdo sdo \
-shift_enable_port se -active hi -clock_port clk -rise -length 10 -name ABS
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Defining Abstract Segments](#)
- [Using Abstract Segments](#)
- [Creating Head, Body, and Tail Segments](#)

Affects this constraint:

[define_dft dft_configuration_mode](#) on page 636
[define_dft scan_chain](#) on page 674

Affects these commands:

[check_dft_rules](#) on page 589
[connect_scan_chains](#) on page 620
[fix_scan_path_inversions](#) on page 707
[report_dft_chains](#) on page 788

Sets these attributes:

[Scan Segment Attributes](#)

define_dft boundary_scan_segment

```
define_dft boundary_scan_segment [-name segment_name]
    {-module subdesign | -instance instance | -libcell cell}
    {-bsdl_string string | -bsdl_file file}
    [-differential_pair {positive_leg_pin negative_leg_pin}]
    [-mode_a mode_a_pin]... [-mode_b mode_b_pin]...
    [-mode_c mode_c_pin]... [-highz highz_pin]
    -tdi tdi_pin -tdo tdo_pin [-clockdr clockdr_pin]
    [-capture_dr capturedr_pin]
    [-updatedr updatedr_pin]
    [-shiftdr shiftdr_pin] [-index bsr_position]
```

Defines a boundary scan segment with its associated pins for connection along the TDI-TDO path in the boundary scan register, connection to the JTAG_Macro, and optionally defines its position in the boundary scan register.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft boundary_scan_segment` command in:

`/designs/top_design/dft/boundary_scan_segments`

Options and Arguments

`-bsdl_file file` Specifies the file containing the BSDL abstract string for the boundary scan segment.

`-bsdl_string string`
 Specifies the BSDL abstract string for the boundary scan segment.

`-capturedr capturedr_pin`
 Specifies the name of the CAPTURE_DR pin on the boundary scan segment.

`-clockdr clockdr_pin`
 Specifies the name of the CLOCK_DR pin on the boundary scan segment.

`-differential_pair {positive_leg_pin negative_leg_pin}`
 Specifies the differential pair in the form of a positive leg pin name and negative leg pin name on the boundary scan segment.

`-highz highz_pin`
 Specifies the name of the HIGHZ pin on the boundary scan segment

Command Reference for Encounter RTL Compiler

Design for Test

<code>-index bsr_position</code>	Specifies the relative position of the boundary scan segment in the BSR. Specify an integer value of zero or greater.
<code>-instance instance</code>	Specifies the instance name of the module for which the boundary scan segment is defined.
<code>-libcell cell</code>	Specifies the library cell for which the boundary scan segment is defined. This option applies to library cells that are implemented as timing models and whose description includes the JTAG Macro related pins specified on the command line when defining the boundary scan segment.
<code>-mode_a mode_a_pin</code>	Specifies the name of the MODE_A pin on the boundary scan segment
<code>-mode_b mode_b_pin</code>	Specifies the name of the MODE_B pin on the boundary scan segment
<code>-mode_c mode_c_pin</code>	Specifies the name of the MODEC pin on the boundary scan segment
<code>-module subdesign</code>	Specifies the subdesign (module) to which the element belongs.
<code>-name segment_name</code>	Defines a name for the boundary scan segment.
<code>-shiftdr shiftdr_pin</code>	Specifies the name of the SHIFT_DR pin on the boundary scan segment.
<code>-tdi tdi_pin</code>	Specifies the name of the TDI pin on the boundary scan segment.
<code>-tdo tdo_pin</code>	Specifies the name of the TDO pin on the boundary scan segment.
<code>-updatedr updatedr_pin</code>	Specifies the name of the UPDATE_DR pin on the boundary scan segment.

Example

- The following example defines an boundary scan segment using a set of differential port pairs.

```
rc:/> define_dft boundary_scan_segment -instance i_pads \
    -bsdl_file pads_bcell.abstract -mode_a MODE_A -mode_b MODE_B -mode_c \
    MODE_C -highz HIGHZ -tdi TDI -tdo TDO -clockdr CLOCKDR -updatedr UPDATEDR \
    -shiftdr SHIFTDR -index 4 \
    -differential_pair {in1 in2} \
    -differential_pair {out1 out2}
```

- The following example defines a boundary scan segment with three mode_a, two mode_b, and one mode_c pins where:

- Each mode_a pin on the boundary-scan segment will be connected to the JTAG_MACRO JTAG_INSTRUCTION_DECODE_MODE_A output pin.
- Each mode_b pin on the boundary-scan segment will be connected to the JTAG_MACRO JTAG_INSTRUCTION_DECODE_MODE_B output pin.
- Each mode_c pin on the boundary-scan segment will be connected to the JTAG_MACRO JTAG_INSTRUCTION_DECODE_MODE_C output pin.

```
rc:/> define_dft boundary_scan_segment-instance i_pads \
    -bsdl_file pads_bcell.abstract -mode_a MODE_A1 -mode_a MODE_A2 \
    -mode_a MODE_A3 -mode_b MODE_B1 -mode_b MODE_B2 -mode_c MODE_C \
    -highz HIGHZ -tdi TDI -tdo TDO -clockdr CLOCKDR \
    -updatedr UPDATEDR -shiftdr SHIFTDR -index 4 -differential_pair {in1 in2} \
    -differential_pair {out1 out2}
```

Related Information

[Defining Boundary Scan Segments in Design for Test in Encounter RTL Compiler](#)

Affects these commands: [insert_dft_boundary_scan](#) on page 717

[write_bsdl](#) on page 808

Sets these attributes: [bcell_segment](#)

[differential](#)

define_dft dft_configuration_mode

```
define_dft dft_configuration_mode
  [-name scan_mode_name]
  [-mode_enable_high test_signal ...]
  [-mode_enable_low test_signal...]
  [-jtag_instruction jtag_instruction]
  [-type {scan |
    wrapper [-usage { extest | intest | mission}] }
  [design]
```

Defines a scan mode. Scan modes can be used to build the top-level scan chains with specific elements in different modes of operation (multi-mode, 1500 wrapper insertion), or when concatenating default scan chains into a single longer scan chain in a different mode of operation.

You can find the objects created by the `define_dft dft_configuration_mode` command in:

```
/designs/top_design/dft/dft_configuration_modes
```

Options and Arguments

design

Specifies the name of the top-level design for which the scan mode is defined. Specify this name if you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-jtag_instruction jtag_instruction_name`

Specifies the JTAG instruction which controls the scan chains in the current mode.

Note: To use this command option you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

`-mode_enable_high test_signal...`

Name of the test signal(s) set to a `logic_1` value that controls scan chains in the current mode.

`-mode_enable_low test_signal...`

Name of the test signal(s) set to a logic_0 value that controls the scan chains in the current mode.

`-name` Name of the scan mode.

`-type {scan | wrapper}`

Specifies the type of the scan mode.

`-usage {extest | intest | mission}`

Specifies the usage of the scan mode for IEEE 1500 core wrapper insertion.

This option is only valid with `-type wrapper`.

Example

The following example defines scan mode `scanModeA`. Test signal `test1` is specified to have an active high logic value and test signal `test2` is specified to have an active low logic value in this mode of operation:

```
define_dft dft_configuration_mode -name scanModeA design \
    -mode_enable_high test1 -mode_enable_low test2
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Defining Scan Chain Configuration Modes](#)
- [JTAG-Controlled Scan Modes](#)
- [Defining the Wrapper Configuration Modes](#)
- [Inserting Core Wrapper Logic](#)

Affects these commands:

[report dft_chains](#) on page 406

[check_dft_rules](#) on page 589

[concat_scan_chains](#) on page 615

[connect_scan_chains](#) on page 620

[define_dft abstract_segment](#) on page 628

[write_atpg](#) on page 805

Command Reference for Encounter RTL Compiler

Design for Test

[write_dft_abstract_model](#) on page 818

[write_et_atpg](#) on page 822

[write_et_bsv](#) on page 830

[write_et_mbist](#) on page 840

[write_et_rrfa](#) on page 844

[write_scandef](#) on page 856

Affects these attributes:

[DFT Configuration Mode Attributes](#)

define_dft domain_macro_parameters

```
define_dft domain_macro_parameters [-name name]
    [-max_num_pulses integer]
    [-counter_length integer | -max_trigger_delay float]
    [-min_target_period float] [-design design]
```

Defines a set of domain macro parameters used to configure and build the OPCG domain macro logic.

You must specify the command with either `-counter_length` or `-max_trigger_delay`.

The command returns the directory path to the `domain_macro_parameter` object that it creates. You can find the objects created by the `define_dft domain_macro_parameters` constraints in:

```
/designs/top/dft/opcg/domain_macro_parameters/
```

Options and Arguments

`-counter_length integer`

Specifies the number of bits in the down counter register.

Specify a number between 4 and 7, or specify 0 if no counter should be inserted.

`-design design`

Specifies the name of the design for which the domain macro parameter set is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

This option is required if multiple designs are loaded.

`-max_number_pulses integer`

Specifies the number of high speed pulses the domain macro should be able to generate. The maximum number you can specify is 8.

Default: 2

`-max_trigger_delay float`

Specifies the time (in picoseconds) after which the first pulse must be issued by the domain macro when it is triggered by the enabled by the TRIGGERRUN output signal generated by the trigger macro.

Command Reference for Encounter RTL Compiler

Design for Test

`-min_target_period float`

Specifies the minimum target period (in picoseconds) at which the domain macro must operate.

Default: 1000

`-name name`

Specifies the `domain_macro_parameter` object name of the domain macro parameter set.

Default: DOMAIN_MACRO_PARAMETER_n

Related Information

[Defining OPCG Domain Macro Parameters in Design for Test in Encounter RTL Compiler](#)

Affects this constraint: [define_dft_opcg_domain](#) on page 662

Affects this command: [iinsert_dft_opcg](#) on page 749

Sets these attributes: [Domain Macro Parameters Attributes](#)

define_dft fixed_segment

```
define_dft fixed_segment [-name segment_name]
{pin|port|instance|segment_name} ...
```

Defines a fixed segment. In a fixed segment, the elements will be connected in the specified order during scan chain connection; they cannot be reordered by a physical scan reordering tool.

A fixed segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the [define_dft scan_chain](#) command
- A tool-created scan chain—created using the [connect_scan_chains](#) command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft fixed_segment` constraints in:

```
/designs/top_design/dft/scan_segments
```

Options and Arguments

{pin|port|instance|segment_name}

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be hierarchical pin, a port, a flip-flop instance, a combinational gate, or a scan segment.

Note: If the segment goes through a multi-input/output combinational gate, you must indicate the scan path through the gate by specifying its input and output pin as two separate consecutive elements.

`-name segment_name` Defines a name for the segment that you can use to reference in the [define_dft scan_chain](#) constraint.

Examples

- The following example defines the fixed segment used in the example for [define_dft scan_chain](#) on page 674.

```
define_dft fixed_segment -name segBody *seq/out_reg_1 *seq/out_reg_3
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example defines a fixed segment that contains two combinational components, four sequential registers, a scan abstract segment, and a combinational component endpoint.

```
define_dft fixed_segment -name fixedSeg \
    i_core/i_anor1/B0 i_core/i_anor1/Y i_core/i_anor2/B0 i_core/i_anor2/Y \
    i_core/i_flop11 i_core/i_flop22 i_core/i_flop33 i_core7i_flop44 \
    absSeg \
    i_core/bufToAnchorSeg/A i_core/bufToAnchorSeg/Y
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Creating Head, Body, and Tail Segments](#)
- [Segmentation Rules](#) in “Exporting the Design”

Affects this constraint: [define_dft scan_chain](#) on page 674

Affects these commands: [connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788

Sets these attributes: [Scan Segment Attributes](#)

define_dft floating_segment

```
define_dft floating_segment [-name segment_name]  
  {pin|port|instance|segment_name} ...
```

Defines a floating segment. In a floating segment, the order of the elements can be changed during scan configuration and by a physical scan reordering tool.

A floating segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the [define_dft_scan_chain](#) command
- A tool-created scan chain—created using the [connect_scan_chains](#) command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft floating_segment` constraints in:

```
/designs/top_design/dft/scan_segments
```

Options and Arguments

`-name segment_name` Defines a name for the segment that you can use to reference in the [define_dft_scan_chain](#) constraint.

`{pin|port|instance|segment_name}`

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be hierarchical pin, a port, a flip-flop instance, or a scan segment.

Examples

- The following example defines the floating segments used in the example for [define_dft_scan_chain](#) on page 674.

```
rc:/designs/test> define_dft floating_segment -name segHead \  
 *seq/out_reg_4 *seq/out_reg_5  
rc:/designs/test> define_dft floating_segment -name segTail *seq/out_reg_0
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Creating Head, Body, and Tail Segments in Design for Test in Encounter RTL Compiler](#)

Affects this constraint: [define_dft_scan_chain](#) on page 674

Affects these commands: [connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788

Sets these attributes: [Scan Segment Attributes](#)

define_dft jtag_instruction

```
define_dft jtag_instruction -name string -opcode string
    [-register string] [-length integer]
    [-register_tdi {pin|port}]
    [-register_tdo {pin|port}]
    [-register_shiftdr {pin|port}]
    [-register_shiftdr_inverted]
    [-register_reset_inverted]
    [-register_capturedr {pin|port}]
    [-register_clockdr {pin|port}]
    [-register_updatedr {pin|port}]
    [-register_tck {pin|port}]
    [-register_reset {pin|port}]
    [-register_runidle {pin|port}]
    [-register_decode {pin|port}]
    [-capture string]
    [-tap_tdo {pin|port}] [-tap_decode {pin|port}]
    [-private] [-design design]
```

Defines a user-defined instruction that is serially loaded into a boundary scan device.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft jtag_instruction` in:

`/designs/top_design/dft/boundary_scan/jtag_instructions/instruction`

Options and Arguments

<code>-capture <i>string</i></code>	Specifies the values that must be captured into a register during the CaptureDR state.
<code>-design <i>design</i></code>	Specifies the name of the design for which the JTAG instruction is defined. If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.
<code>-length <i>integer</i></code>	Specifies the length of the custom test data register (TDR).
<code>-name <i>string</i></code>	Specifies the name of the user-defined instruction.
<code>-opcode <i>string</i></code>	Specifies the binary code for this instruction.
<code>-private</code>	Specifies that the defined instruction is private.
<code>-register <i>string</i></code>	Specifies the name of the custom test data register (TDR).

Command Reference for Encounter RTL Compiler

Design for Test

-register_capturedr {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_CAPTUREDR` pin on the `JTAG_MACRO` subdesign.

-register_clockdr {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_CLOCKDR` pin on the `JTAG_MACRO` subdesign .

-register_decode {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_INSTRUCTION_DECODE_instruction` pin on the `JTAG_MACRO` subdesign, where *instruction* is the name that you specified through the `-name` option.

-register_reset {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_RESET` pin on the `JTAG_MACRO` subdesign.

-register_reset_inverted

Specifies that the `JTAG_RESET` pin of the custom test register (TDR) has an active low polarity.

-register_runidle {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_RUNIDLE` pin on the `JTAG_MACRO` subdesign.

-register_shiftdr {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the `JTAG_SHIFTDR` pin on the `JTAG_MACRO` subdesign.

-register_shiftdr_inverted

Specifies that the `JTAG_SHIFTDR` pin of the custom test register (TDR) has an active low polarity.

Command Reference for Encounter RTL Compiler

Design for Test

-register_tck {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the JTAG_TCK pin on the JTAG_MACRO subdesign.

-register_tdi {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the JTAG_TDI pin on the JTAG_MACRO subdesign.

-register_tdo {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the JTAG_register_TDO pin on the JTAG_MACRO subdesign, where *register* is the name that you specified through the -register option

-register_updatedr {pin|port}

Specifies the name of the pin on the custom test data register (TDR) that must be connected to the JTAG_UPDATEDR pin on the JTAG_MACRO subdesign.

-tap_decode {pin|port}

Specifies the name of the instruction-specific decode pin that must be created on the JTAG_MACRO subdesign.

-tap_tdo {pin|port}

Specifies the name of the instruction-specific test data output (TDO) pin that must be created on the JTAG_MACRO subdesign.

Example

- The following example defines a private instruction PROGRAM_TCB for custom test data register TCB_REG that has a length of 8 bits. The opcode for the instruction is 0101.

```
define_dft jtag_instruction -name PROGRAM_TCB -opcode 0101 -register TCB_REG  
-length 8 -private
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Defining the Instructions](#)
- [Inserting a JTAG Macro](#)
- [Inserting Memory Built-In-Self-Test Logic](#)
- [JTAG-Controlled Scan Modes](#)

Affects these commands: [insert_dft_boundary_scan](#) on page 717

[insert_dft_jtag_macro](#) on page 736

[insert_dft_mbist](#) on page 743

Related command: [define_dft_jtag_instruction_register](#) on page 649

Sets these attributes: [JTAG Instruction Attributes](#)

define_dft jtag_instruction_register

```
define_dft jtag_instruction_register  
  -name string  
  [-length integer] [-capture string]  
  [-design design]
```

Customizes the instruction register to allow adding user-defined instructions.

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft jtag_instruction_register` in:

```
/designs/top_design/dft/boundary_scan/register_name
```

Options and Arguments

<code>-capture <i>string</i></code>	Specifies the capture value of the instruction register. According to the IEEE 1149.1 standard the last two bits of the capture value must be 01. <i>Default:</i> 01
<code>-design <i>design</i></code>	Specifies the name of the design for which the instruction register is customized. If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.
<code>-length <i>integer</i></code>	Specifies the length of the instruction register. The length of the register is determined by the number of user-defined instructions that you want to add. If <i>n</i> is the number of bits in the instruction register, a total of 2^n instructions can be defined including the four mandatory instructions. <i>Default:</i> 2
<code>-name <i>string</i></code>	Specifies the name of the user-defined instruction register.

Examples

- The following example defines instruction register `INSTR_REGISTER` with length 3. A register of length 3 allows you to create 2^3 instructions, or four user-defined instructions besides the four mandatory instructions.

```
define_dft jtag_instruction_register -name INSTR_REGISTER -length 3
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Defining the Instruction Register](#)
- [Inserting a JTAG Macro](#)
- [Inserting Memory Built-In-Self-Test Logic](#)
- [JTAG-Controlled Scan Modes](#)

Related commands: [define_dft_jtag_instruction](#) on page 645

Affects these commands: [insert_dft_boundary_scan](#) on page 717

[insert_dft_jtag_macro](#) on page 736

[insert_dft_mbist](#) on page 743

Related attributes: [JTAG Instruction Register Attributes](#)

define_dft jtag_macro

```
define_dft jtag_macro
    [-module subdesign] [-libcell libcell]
    [-instance instance]
    [-bsr_shiftdr {pin|port|subport}]
    [-bsr_clockdr {pin|port|subport}]
    [-bsr_updatedr {pin|port|subport}]
    [-reset {pin|port|subport}]
    [-runidle {pin|port|subport}]
    [-shiftdr {pin|port|subport}]
    [-clockdr {pin|port|subport}]
    [-updatedr {pin|port|subport}]
    [-capturedr {pin|port|subport}]
    [-mode_a {pin|port|subport}]
    [-mode_b {pin|port|subport}]
    [-mode_c {pin|port|subport}]
    -tdi {pin|port|subport} -tdo {pin|port|subport}
    -tck {pin|port|subport} -tms {pin|port|subport}
    [-trst pin/port/subport]
    [-boundary_tdo {pin|port|subport}]
    [-tdo_enable {pin|port|subport}]
    [-highz {pin|port|subport}]
    [-dot6_acdcsel {pin|port|subport}]
    [-dot6_acpulse {pin|port|subport}]
    [-dot6_preset_clock {pin|port|subport}]
    [-dot6_trcell_enable {pin|port|subport}]
    [-por {pin|port|subport}] [-name string]
```

Identifies a pre-instantiated JTAG Macro.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-boundary_tdo {pin|port|subport}

Specifies the boundary register TDO output pin on the JTAG macro.

-bsr_clockdr {pin|port|subport}

Specifies the clock data register (CLOCKDR) output pin for the boundary-scan register.

Command Reference for Encounter RTL Compiler

Design for Test

-bsr_shiftdr {pin|port|subport}
Specifies the shift data register (SHIFTDR) output pin for the boundary-scan register.

-bsr_updatedr {pin|port|subport}
Specifies the update data register (UPDATEDR) output pin for the boundary-scan register.

-clockdr {pin|port|subport}
Specifies the clock data register (CLOCKDR) output pin custom test data register.

-capturedr {pin|port|subport}
Specifies the capture data register (CAPTUREDR) output pin custom test data register.

-dot6_acdcsel {pin|port|subport}
Specifies the logical OR of the decoded EXTEST_PULSE and EXTEST_TRAIN instructions. The tool connects this signal to the AC Mode pin of all test receivers when it inserts the boundary scan logic. This signal also controls the multiplexer which is added to the output boundary scan cells.

-dot6_acpulse {pin|port|subport}
Specifies the AC test signal output of the JTAG macro.

-dot6_preset_clock {pin|port|subport}
Specifies the preset_clock output pin that provides a positive-active edge-sensitive clock signal to test receivers that have edge-sensitive initialization.

-dot6_trcell_enable {pin|port|subport}
Specifies the logical OR of EXTEST, EXTEST_PULSE and EXTEST_TRAIN used to enable the test receiver cells.

-highz {pin|port|subport}
Specifies the HIGHZ output pin to place the I/O pads in their HIGHZ state.

-instance *instance* Specifies the path name of the JTAG_MACRO instance.

-libcell *cell* Specifies the library cell for the JTAG_MACRO instance.

Command Reference for Encounter RTL Compiler

Design for Test

- mode_a {pin|port|subport}
 - Specifies the mode_a output pin to configure boundary cells in the boundary-scan register.
- mode_b {pin|port|subport}
 - Specifies the mode_b output pin to configure boundary cells in the boundary-scan register.
- mode_c {pin|port|subport}
 - Specifies the mode_c output pin to configure boundary cells in the boundary-scan register.
- module subdesign
 - Specifies the subdesign (module) name of the JTAG Macro instance.
- name subdesign
 - Specifies the name of the JTAG_MACRO. If not specified, an object will be added under the boundary_scan/jtag_macros vdir with a default name of jtag_macro_n.
- por {pin|port|subport}
 - Specifies the power-on reset input pin on the JTAG macro.
- reset {pin|port|subport}
 - Specifies the reset output pin indicating that the JTAG macro is in the Test-Logic-Reset state.
- runidle {pin|port|subport}
 - Specifies the JTAG_RUNIDLE output pin indicating that the JTAG macro is in the Run-Test-Idle state.
- shiftdr {pin|port|subport}
 - Specifies the shift data register (SHIFTDR) output pin custom test data register.
- tck {pin|port|subport}
 - Specifies the TAP controller TCK input pin on the JTAG macro.
- tdi {pin|port|subport}
 - Specifies the TAP controller TDI input pin on the JTAG macro.
- tdo {pin|port|subport}
 - Specifies the TAP controller TDO output pin on the JTAG macro.

Command Reference for Encounter RTL Compiler

Design for Test

-tdo_enable {pin|port|subport}

Specifies the enable output pin which drives the JTAG TDO output enable pin.

-tms {pin|port|subport}

Specifies the TAP controller TMS input pin on the JTAG macro.

-trst {pin|port|subport}

Specifies the TAP controller TRST input pin on the JTAG macro.

-updatedr {pin|port|subport}

Specifies the update data register (UPDATEDR) output pin custom test data register.

Examples

- The following example defines a third party TAP controller with a specified instance location.

```
rc:/> define_dft jtag_macro -instance user_defined_jtag \
-highz MY_JTAG_INSTRUCTION_DECODE_CTRL_HIGHZ \
-bsr_clockdr MY_JTAG_BOUNDARY_CLOCKDR -bsr_shiftdr MY_JTAG_BOUNDARY_SHIFTDR \
-bsr_updatedr MY_JTAG_BOUNDARY_UPDATEDR \
-mode_a MY_JTAG_INSTRUCTION_DECODE_MODE_A \
-mode_b MY_JTAG_INSTRUCTION_DECODE_MODE_B \
-mode_c MY_JTAG_INSTRUCTION_DECODE_MODE_C -tdi MY_JTAG_TDI -tdo MY_JTAG_TDO \
-tms MY_JTAG_TMS -tck MY_JTAG_TCK -trst MY_JTAG_TRST \
-tdo_enable MY_JTAG_ENABLE_TDO -boundary_tdo MY_JTAG_BOUNDARY_TDO \
-por MY_JTAG_POR -name JM1
```

- The following example defines a third party TAP controller without a TRST input pin.

```
rc:/> define_dft jtag_macro -module MY_JTAG_MACRO \
-highz MY_JTAG_INSTRUCTION_DECODE_CTRL_HIGHZ \
-bsr_clockdr MY_JTAG_BOUNDARY_CLOCKDR -bsr_shiftdr MY_JTAG_BOUNDARY_SHIFTDR \
-bsr_updatedr MY_JTAG_BOUNDARY_UPDATEDR \
-mode_a MY_JTAG_INSTRUCTION_DECODE_MODE_A \
-mode_b MY_JTAG_INSTRUCTION_DECODE_MODE_B \
-mode_c MY_JTAG_INSTRUCTION_DECODE_MODE_C -tdi MY_JTAG_TDI -tdo MY_JTAG_TDO \
-tms MY_JTAG_TMS -tck MY_JTAG_TCK -tdo_enable MY_JTAG_ENABLE_TDO \
-boundary_tdo MY_JTAG_BOUNDARY_TDO -por MY_JTAG_POR -name JM1
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Defining a Pre-Existing JTAG Macro](#)
- [JTAG-Controlled Scan Modes](#)

Sets these attributes: [JTAG Macro Attributes](#)

Related command: [insert_dft_jtag_macro](#) on page 736

define_dft mbist_clock

```
define_dft mbist_clock -name mbist_clock
  [-design design] [-internal_clock_source]
  -period integer [-divide_period integer]
  [-hookup_pin pin] [-hookup_period string]
  [-hookup_polarity {non_inverted | inverted}] port
```

Defines an MBIST clock and associates a clock waveform with the clock. The clock waveform can be different from the system clocks.

You must define all MBIST clocks that are referenced in the MBIST configuration file to enable the DFT rules checking associated with MBIST insertion in the design.

Note: The system function for the boundary cell associated with the external clock source must be `clock`. For more information, refer to [Custom Boundary-Scan Cells](#) in the *Design for Test in Encounter RTL Compiler*

The command returns the directory path to the `mbist` object that it creates. You can find the objects created by the `define_dft mbist_clock` constraints in:

```
/designs/design/dft/mbist/mbist_clocks
```

Options and Arguments

`-design design` Specifies the name of the design for which the MBIST clock is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-divide_period integer`

Together with the `-period` option, determines the MBIST clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 1

`-hookup_period integer`

Specifies the period interval at the hookup pin. The hookup period is specified in picoseconds.

Default: value of `-period` option

`-hookup_pin pin`

Specifies the core-side hookup pin to be used by the `insert_dft mbist` command to make the MBIST clock connection.

Command Reference for Encounter RTL Compiler

Design for Test

Note: When you specify this option, the RC-DFT engine does not validate the controllability of any logic between the top-level MBIST clock port and its designated hookup pin under test-mode setup.

`-hookup_polarity {non_inverted | inverted}`

Specifies the polarity of the MBIST clock signal at the core-side hookup pin relative to the specified port.

Default: non_inverted

`-internal_clock_source`

Specifies that the MBIST clock is either driven by a an internal clock source internal to the design or by an external clock source, design port, that has a free-running clock applied at the tester.

`-name mbist_clock`

Specifies the name of the MBIST clock that is being defined.

Each clock object in your design must have a unique name. If you define a new MBIST clock with the same name as an existing clock, an error message will be issued.

Note: The clock name is referenced within the MBIST configuration file to access this clock object. The clock name also allows you to search for the clock later (through the `find` command) or to recognize it in reports.

`-period integer`

Together with the `-divide_period` option, determines the clock period interval at the specified port. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 50000 (20 MHz test clock)

`port`

Specifies the MBIST clock input port, or the *test time control* port in case of an internal MBIST clock.

The test time control port is any port on the design which can toggle during memory test to control the number of test cycles executed to ensure the internally clocked MBIST test completes.

Example

- The following example defines three MBIST clocks, the second one needing a hookup pin to avoid a PLL and the third one specifying an internal clock source.

```
define_dft mbist_clock -name CLK1X -period 20000 CLK1
define_dft mbist_clock -name CLK2X -period 20000 -hookup_pin PLLOUTA \
    -hookup_period 10000 CLK2
define_dft mbist_clock -name CLK3I -period 30000 -hookup_pin OSCOUT \
    -internal_clock_source CLK2
```

For the MBIST clock object `CLK2X` there is a clock frequency multiplication factor of 2 through the PLL and it is in phase with the port `CLK2`. For the MBIST clock object `CLK3I` the clock is driven by an on-chip oscillator which reuses design port `CLK2` as the test time control port.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [MBIST Clocking](#) in “Inserting Memory Built-In-Self-Test Logic”
- [Design Flows](#) in “Inserting Memory Built-In-Self-Test Logic”

Affects these commands: [check_mbist_rules](#) on page 595

[insert_dft mbist](#) on page 743

Sets these attributes: [MBIST Clock Attributes](#)

define_dft mbist_direct_access

```
define_dft mbist_direct_access -function string
    -active {low | high}
    [-mtclk] {port | pin}
```

Defines MBIST direct access interface pins or ports which can be used as an alternative access mechanism for MBIST. This method can be used as a supplement to or replacement for the JTAG interface for controlling MBIST.

All signals are level-sensitive. The patterns that you generate by running the `create_embedded_test` command in Encounter Test, properly stimulate and monitor the defined signals if they are directly accessible from design ports.

Execution scheduling options are limited to all MBIST controllers running in parallel or serially and the devices assigned to each engine running in parallel or serially.

Options and Arguments

- active {low|high} Specifies the active value of the signal.
- function *string* Defines the functionality of the direct access pin. The option can have one of the following values:
 - `burnin_run`
This signal must be held inactive for a minimum of three MBIST clock periods at the start of the test sequence. Once activated, the `burnin_run` signal causes the MBIST operations to execute continuously within a loop until the signal is deactivated. The results of the operations are made visible on the `monitor` during the course of the execution.
 - `device_schedule_serial`
Causes the execution of devices within each MBIST controller to occur serially when the signal is active. This signal must remain stable throughout the test sequence.
The execution occurs in parallel when the signal is inactive or undefined.

Command Reference for Encounter RTL Compiler

Design for Test

- `engine_schedule_serial`

Causes the execution of MBIST controllers in the design to occur serially. This signal must remain stable throughout the test sequence.

The execution occurs in parallel when the signal is inactive or undefined.

- `monitor`

This signal is the logical AND of all MBIST controller done indications and inverted summary failure indications during a `poweron_run` operation. During a `burnin_run` operation, the signal is the logical AND of all MBIST inverted summary failure indications.

Note: This function can also be used by a JTAG controller to monitor MBIST operations. You can specify it as the only direct access function in cases where the JTAG is the only access mechanism used for MBIST. In these circumstances, the `monitor` connection must not be shared and must not require any gating condition to enable observation.

- `poweron_run`

This signal must be held inactive for a minimum of three MBIST clock periods at the start of the test sequence. When activated, a single execution of the MBIST operations occurs and the results of the operations are made visible on the `monitor` at completion.

`-mtclk`

Specifies whether the alternate `mtclk` clock input is used during direct access as applied to `burnin_run` or `poweron_run` functions.

`{pin | port}`

Specifies the source pin or port of the signal.

A pin cannot have the `inout` direction if is used to monitor.

A port cannot have the `inout` direction if is used to monitor and you use the MBIST block insertion flow.

Any pin or port can be shared for functional uses provided the MBIST DFT configuration mode enables memory BIST usage during MBIST operations.

A pin specification for the `monitor` function implies that an internal monitor point is requested. The tool will not perform a forward trace to look for a port.

Example

The following example defines a power-on activated memory BIST with monitor and default execution schedule of parallel engine, parallel device.

```
define_dft mbist_direct_access -function poweron_run -active high \
    ports_in/poweron_mbist
define_dft mbist_direct_access -function monitor -active low \
    ports_out/DFT_scan_out[7]
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [RTL Compiler Prerequisites](#) in “Inserting Memory Built-In-Self-Test Logic”
- [Design Flows](#) in “Inserting Memory Built-In-Self-Test Logic”

Affects these commands: [check_mbist_rules](#) on page 595

[insert_dft_mbist](#) on page 743

Sets these attributes: [Direct Access Function Attributes](#)

define_dft opcg_domain

```
define_dft opcg_domain [-name name]
    -osc_source osc_source
    -domain_macro_parameter domain_macro_parameter
    -opcg_trigger opcg_trigger
    -location {subport | port | pin}
    -min_domain_period float
    [-divide_by integer] [-design design]
```

Defines an OPCG (clock) domain.

The command returns the directory path to the `opcg_domain` object that it creates. You can find the objects created by the `define_dft opcg_domain` constraints in:

```
/designs/top/dft/opcg/opcg_domains/
```

Options and Arguments

<code>-design design</code>	Specifies the name of the design for which the domain is defined. If you omit the design name, the top-level design of the current directory of the design hierarchy is used. This option is required if multiple designs are loaded.
<code>-divide_by integer</code>	Specifies to build an internal clock divider circuit which divides the oscillator source frequency by the specified number.
<code>-domain_macro_parameter domain_macro_parameter</code>	Specifies the domain macro parameter set to be used for the OPCG domain. The domain macro parameter set must have been previously defined by the <code>define_dft domain_macro_parameters</code> command.
<code>-location {subport port pin}</code>	Specifies where to insert the domain macro. Specify an existing top-level port, subport, or hierarchical pin name.
<code>-min_domain_period float</code>	Specifies the minimum period (in picoseconds) at which the OPCG domain is assumed to operate.

Note: The frequency at which the domain is supposed to operate (specified using this option) must be greater than the minimum frequency of the oscillator source, specified using the `-max_output_period` of the `define_dft osc_source` command.

`-name name` Specifies the `opcg_domain` object name of the OPCG domain.

Note: The name allows you to search for the object later (through the `find` command) or to recognize it in reports.

Default: `OPCG_DOMAIN_n`

`-opcg_trigger opcg_trigger`

Specifies the trigger signal that will enable high speed pulses to be generated by the domain macros controlled by this trigger.

The trigger signal must have been previously defined with the `define_dft opcg_trigger` constraint.

`-osc_source osc_source`

Specifies the oscillator source to be used for the OPCG domain.

The oscillator source must have been previously defined by a `define_dft osc_source` constraint.

Examples

- The following command inserts the domain macro after output pin Y of buffer instance i_buf1.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_buf1/Y -min_domain_period 1000 -divide_by 6
```

- The following command inserts the domain macro before input pin A of buffer instance i_buf1.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_buf1/A -min_domain_period 1000 -divide_by 6
```

- The following command inserts the domain macro after pin out1 of subdesign instance i_core.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_core/out1 -min_domain_period 1000 -divide_by 6
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following command inserts the domain macro within the subdesign for instance `i_core` prior to output pin `out1`.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_core/subports_out/out1 -min_domain_period 1000 -divide_by 6
```

- The following command inserts the domain macro before the `in1` pin of subdesign instance `i_core`.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_core/in1 -min_domain_period 1000 -divide_by 6
```

- The following command inserts the domain macro within the subdesign for instance `i_core` after the input pin `in1`.

```
define_dft opcg_domain -name OD1 -osc_source OSC1 \
-domain_macro_parameter DOMAIN_MACRO_PARAMETER_1 -opcg_trigger TRIGGER1 \
-location i_core/subports_in/in1 -min_domain_period 1000 -divide_by 6
```

Related Information

[Defining OPCG Clock Domains in Design for Test in Encounter RTL Compiler](#)

Affected by these constraints: [define_dft domain macro parameters](#) on page 639
 [define_dft opcg trigger](#) on page 667
 [define_dft osc source](#) on page 669

Affects this command: [insert_dft opcg](#) on page 749

Sets these attributes: [OPCG Domain Attributes](#)

define_dft opcg_mode

```
define_dft opcg_mode [-name name]  
    [-mode_init file]  
    [-jtag_controlled]  
    [-osc_source_parameters string]  
    [-osc_source_parameters string]...  
    [-design design]
```

Defines an OPCG mode for Encounter Test ATPG.

The command returns the directory path to the `opcg_mode` object that it creates. You can find the objects created by the `define_dft opcg_mode` constraints in:

```
/designs/top/dft/opcg/opcg_modes/
```

Options and Arguments

<code>-design <i>design</i></code>	Specifies the name of the design for which the OPCG mode is defined. This option is only required if multiple designs are loaded. If there is only one top-level design, you can omit the design name. In this case, the tool will use the top-level design of the design hierarchy.
<code>-jtag_controlled</code>	Specifies whether a JTAG instruction is used to lock the PLLs for OPCG operation.
<code>-mode_init <i>file</i></code>	Specifies the name of file containing the OPCG mode initialization sequence that ensures that PLLs are properly initialized and locked on the input oscillators.
<code>-name <i>name</i></code>	Specifies the <code>opcg_mode</code> object name of the OPCG mode. <i>Default:</i> <code>DFT_prefix_opcg_mode_n</code>
<code>-osc_source_parameters <i>string</i></code>	Specifies the oscillator source parameters specific for this OPCG mode. Use the following format: <code>{<i>name</i> <i>osc_source_output_period</i> <i>ref_clock_period</i>}</code> where <i>name</i> is the name of the <code>osc_source</code> object.

Example

The following command defines an OPCG mode whose name defaults to DFT_opcg_mode_1.

```
define_dft opcg_mode -osc_source_parameters {OSC1 52.5 270.3} \
           -osc_source_parameters {OSC2 52 290} -mode_init opcgMode1.mode_init
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Defining the OPCG Mode](#)
- [Writing the Scripts and Setup Files to Perform ATPG](#)
- [Generate Files for ATPG and Simulation](#)

Affects this command: [write_et_atpg](#) on page 822

Related constraint: [define_dft osc_source](#) on page 669

Sets these attributes: [OPCG Mode Attributes](#)

[Osc Source Reference Attributes](#)

define_dft opcg_trigger

```
define_dft opcg_trigger [-name name]
    -active {low|high}
    -osc_source osc_source
    {pin|port} [-create_port]
    [-design design]
```

Defines the trigger signal that will enable the trigger macro associated with the specified oscillator source.

Note: Although each oscillator source requires a unique trigger macro, multiple trigger macros can share the same trigger signal pin.

The command returns the directory path to the `opcg_trigger` object that it creates. You can find the objects created by the `define_dft opcg_trigger` constraints in:

```
/designs/top/dft/opcg/opcg_triggers/
```

Options and Arguments

`-active {low | high}`

Specifies the active value for the OPCG trigger signal.

`-create_port`

Specifies whether to create the port if it does not exist.

`-design design`

Specifies the name of the design for which the trigger enable is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

This option is required if multiple designs are loaded.

`-name name`

Specifies the `opcg_trigger` object name of the OPCG trigger signal.

Default: OPCG_TRIGGER_n

`-osc_source osc_source`

Specifies the object name of the oscillator source whose trigger macro is enabled by the defined signal.

The oscillator source must have been defined with `define_dft osc_source` command.

`{pin|port}`

Specifies the driving (input) pin or port for the OPCG trigger signal.

Note: If multiple designs are loaded and you did no specify the -design option, you can also specify the full path to the driver.

Example

The following command defines the opcg_trigger object associated with oscillator source OSC on port GO . This port will be created if it does not exist.

```
define_dft opcg_trigger -name TRIGGER -create_port -active low -osc_source OSC GO
```

Related Information

[Defining OPCG Triggers in Design for Test in Encounter RTL Compiler](#)

Affects this command: [insert dft opcg on page 749](#)

Related constraints: [define dft osc source on page 669](#)

[define dft opcg domain on page 662](#)

[define dft opcg mode on page 665](#)

Sets these attributes: [OPCG Trigger Attributes](#)

define_dft osc_source

```
define_dft osc_source [-name osc_source]
    -ref_clock_pin {pin|port}
    -min_input_period integer
    -max_input_period integer
    -min_output_period integer
    -max_output_period integer
    pin
    [-design design]
```

Defines an oscillator source on the output pin of a PLL instance that will drive the OPCG logic.

The command returns the directory path to the `osc_source` object that it creates. You can find the objects created by the `define_dft osc_source` constraints in:

```
/designs/top/dft/opcg/osc_sources/
```

Options and Arguments

- | | |
|--|---|
| <code>-design <i>design</i></code> | Specifies the name of the design for which the oscillator clock is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

This option is required if multiple designs are loaded. |
| <code>-max_input_period <i>float</i></code> | Specifies the maximum period of the reference (input) clock.
Specify the clock period in picoseconds. The value of this option must be larger than the value of the <code>-min_input_period</code> option. |
| <code>-max_output_period <i>float</i></code> | Specifies the maximum period of the generated (output) clock.
Specify the clock period in picoseconds. The value of this option must be larger than the value of the <code>-min_output_period</code> option. |
| <code>-min_input_period <i>float</i></code> | Specifies the minimum period of the reference (input) clock.
Specify the clock period in picoseconds. |

<code>-min_output_period float</code>	Specifies the minimum period of the generated (output) clock. Specify the clock period in picoseconds.
<code>-name osc_source</code>	Specifies the name of the <code>osc_source</code> object for the oscillator source.
	<i>Default:</i> OSC_SOURCE_n
<code>pin</code>	Specifies the output pin of the PLL.
<code>-ref_clock_pin pin</code>	Specifies the reference (input) clock pin or port for the PLL.

Example

The following command defines an oscillator source for instance `instPLL1` whose reference (input) clock has a period between 20000 and 4000 ps (or clock frequency between 50 and 250 MHz) and whose output clock has a period between 2000 and 500 ps (or a clock frequency between 500 and 2000 MHz).

```
define_dft osc_source -name instPLL1
  -ref_clock_pin REFCLKPORT1 \
  -min_input_period 4000 -max_input_period 20000
  -min_output_period 500 -max_output_period 2000 \
  PLL1/z1
```

Related Information

[Defining the Oscillator Sources in Design for Test in Encounter RTL Compiler](#)

Affects these commands

[define_dft opcg_domain](#) on page 662
[define_dft opcg_mode](#) on page 665
[define_dft opcg_trigger](#) on page 667
[insert_dft opcg](#) on page 749

Sets these attributes:

[Osc Source Attributes](#)

define_dft preserved_segment

```
define_dft preserved_segment [-name segment_name]
  {instance|segment_name}... [-sdi pin] [-sdo pin]
  | -analyze -sdi {pin|port} -sdo {pin|port} }
  [ -connected_shift_enable
  | [-connected_scan_clock_a] [-connected_scan_clock_b] ]
  [-allow_reordering]
```

Defines a preserved segment. In a preserved segment, the elements of the mapped segment are already connected in the specified order, and they cannot be reordered by a physical scan reordering tool.

A preserved segment is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the [define_dft scan_chain](#) command
- A tool-created scan chain—created using the [connect_scan_chains](#) command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft preserved_segment` constraints in:

`/designs/top_design/dft/scan_segments`

Options and Arguments

<code>-allow_reordering</code>	Indicates whether the order of the elements can be changed during scan configuration and by a physical scan reordering tool.
<code>-analyze</code>	Analyzes the connectivity of the scan segment, and returns the elements of the segment given its endpoints.
<code>-connected_scan_clock_a</code> (<code>-connected_scan_clock_b</code>)	<p>Indicates that the <code>scan_clock_a</code> (<code>scan_clock_b</code>) port of the module boundary is driven by external logic (preconnected). The external logic connected to the <code>scan_clock_a</code> (<code>scan_clock_b</code>) pin of the module will not be modified by the scan configuration engine.</p> <p>This option applies only for the clocked LSSD scan style.</p>

Command Reference for Encounter RTL Compiler

Design for Test

`-connected_shift_enable`

Indicates that the shift enable port of the module boundary is driven by external logic (preconnected) or that the shift enable signal is internally generated within the module boundary. In either case, the external logic connected to the shift enable pin of the module, or the internal logic driving the shift enable pins of the flip-flops in the module will not be modified by the scan configuration engine.

`{instance|segment_name}`

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be a flip-flop instance, a combinational instance, or a scan segment.

Additionally, the hierarchical scan data input and output pins of the segment can be specified using the `-sdi` and `-sdo` options respectively. If the hierarchical SDI and SDO pins of the segment are both at the boundary of the same lower module, the RC-DFT engine also traces the shift-enable signal back from the scan registers in the segment to the same module boundary. It hooks up the shift-enable signal at the module boundary whenever applicable (only if you did not specify the `-connected_shift_enable` option).

Specifies an element in the scan segment being defined. List the elements in the order they should appear in the scan chain (in shift order, that is, left-most corresponds to first bit shifted-in). An element can be a flip-flop instance, a buffer, an inverter, a (hierachal) pin, or a scan segment.

`-name segment_name` Defines a name for the segment that you can use to reference in the [define_dft_scan_chain](#) constraint.

`-sdi (-sdo)` Specifies the scan data input (scan data output) of the segment. Specify a hierarchical pin name or port.

Examples

- The following example defines a pre-existing segment in instance `u_a` using its scan data input and output pins.

```
rc:/> define_dft preserved_segment -name segmenta -analyze \
-sdi */u_a/SIa -sdo */u_a/SOa
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example defines a pre-existing segment by specifying its hierarchical scan data input and output pins, and its elements consisting of two combinational components, four sequential registers, a scan abstract segment, and a combinational component endpoint.

```
define_dft preserved_segment -name preservedSeg \
    -sdi i_core/i_anor1/B0 -sdo i_core/bufToAnchorSeg/Y \
    i_core/i_anor1 i_core/i_anor2 \
    i_core/i_flop11 i_core/i_flop22 i_core/i_flop33 i_core/i_flop44 \
    absSeg \
    i_core/bufToAnchorSeg
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Handling Preexisting Scan Segments](#)
- [Creating Head, Body, and Tail Segments](#)
- [Segmentation Rules](#) in “Exporting the Design”

Affects this constraint: [define_dft scan_chain](#) on page 674

Affects these commands: [connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788

Sets these attributes: [Scan Segment Attributes](#)

define_dft scan_chain

```
define_dft scan_chain [-name name]
  {[-sdi sdi -sdo sdo [-create_ports]
   {-shared_output [-shared_select test_signal] |
    -non_shared_output}
   [-hookup_pin_sdi pin] [-hookup_pin_sdo pin]
   [-shift_enable test_signal]
   [-head segment] [-tail segment] [-body segment]
   [-complete | -max_length integer]
   [-domain test_clock_domain [-edge {rise|fall}]]]
   [-terminal_lockup {level_sensitive|edge_sensitive}]
   [-configure_pad {tm_signal | se_signal} ]
   |-analyze -sdo sdo [-sdi sdi] [-dont_overlay]
   {-shared_out | -non_shared_out} }
```

Creates a scan chain or analyzes an existing chain with the specified input and output scan data ports.

If you created a scan chain, the command returns the directory path to the `scan_chain` object that it creates. For newly created scan chains you can find the objects created by the `define_dft scan_chain` constraints in:

```
/designs/top_design/dft/scan_chains
```

If you successfully analyzed an existing scan chain, the command returns the directory path to the `actual_scan_chain` object that it creates. For successfully analyzed scan chains, you can find the objects created by the `define_dft scan_chain` constraints in:

```
/designs/top_design/dft/report/actual_scan_chains
```

Options and Arguments

<code>-analyze</code>	Analyzes the connectivity of an existing top-level scan chain in a structural netlist compiled in a previous RC session. You must at least specify the scan data output pin and optionally the scan data input pin to identify the chain.
<code>-body segment</code>	Indicates that the specified segment (an ordered set of scan flip-flops) is part of the body of the scan chain. The segment must have been previously defined with a <code>define_dft xxx_segment</code> constraint, where <code>xxx</code> is either <code>abstract</code> , <code>fixed</code> , <code>floating</code> , <code>preserved</code> , or <code>shift_register</code> .
<code>-complete</code>	Specifies that the defined chain is complete and no other flip-flops should be added to the chain.

Command Reference for Encounter RTL Compiler

Design for Test

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal (test mode or shift enable signal) that the RC-DFT engine must use if it needs to configure the pad connected to the scan data input or output signal to control the data direction during test mode.



Tip

If the scan data input and output pin are shared with functional pins, you should use the shift enable test signal to configure pads. This will allow the pads to shift-in and shift-out data when shift-enable signal is active (scan-shift mode), and will allow the pads to operate in functional mode when shift-enable signal is inactive (capture mode).

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_ports`

Specifies whether to create the scan data input and output ports if they do not exist.

If you do not specify the scan data input or output signals using the `-sdi` and `-sdo` options, the ports can be created and named as `prefix_sdi_num` and `prefix_sdo_num`, where `prefix` is the value of the `dft_prefix` attribute.

`-domain test_clock_domain`

Specifies the DFT clock domain to associate with the scan chain. This clock domain must have been previously identified by the `check_dft_rules` command or defined with the `define_dft test_clock` constraint.

If you omit this option, and segments have been defined for the chain, the scan chain is automatically associated with the appropriate DFT clock domain. In the absence of segments, the scan chain can be assigned to any DFT clock domain.

Note: This option only applies to the muxed scan style.

Command Reference for Encounter RTL Compiler

Design for Test

<code>-dont_overlay</code>	Prevents that RTL Compiler reassociates (or overlays) user-defined segments of type preserve or fixed to its analyzed scan chains. Consequently, the segments are not re-established as a fixed entity (and hence are reorderable) when determining how to partition the chains for physical-based reordering. If these segments include multi-input combinational logic gates in the scan data path, the <code>write_scandef</code> command uses these gates to partition the analyzed scan chains into n-reorderable segments (referred to as scanDEF chains). The register before a multi-input combinational logic gate becomes the STOP point for one scanDEF chain, while the register after a combinational gate becomes the START point of another scanDEF chain.
<code>-edge {rise fall}</code>	<p>Specifies whether to use the falling or rising edge of the test clocks in the specified DFT clock domain. You can specify this option only if you specified <code>-domain</code>.</p> <p>If you omit this option, the scan flip-flops triggered by the different active edges of the test clocks will be placed on their own scan chain.</p> <p>This option is ignored if you enabled the <code>dft_mix_clock_edges_in_scan_chains</code> attribute.</p> <p>Note: This option only applies to the muxed scan style.</p>
<code>-head <i>segment</i></code>	Indicates that the specified segment (an ordered set of scan flip-flops) must be placed at the head (closest to the scan data input) of the scan chain. The segment must have been previously defined with a <code>define_dft xxx_segment</code> constraint, where <code>xxx</code> is either <code>abstract</code> , <code>fixed</code> , <code>floating</code> , <code>preserved</code> , or <code>shift_register</code> .
<code>-hookup_pin_sdi <i>pin</i></code>	Specifies the core-side hookup pin to be used for the scan data input signal during scan chain connection.
	<p>Note: When you specify this option, the RC-DFT engine does not validate the control ability of any logic between the top-level scan data input signal and its designated hookup pin under test-mode setup.</p>
<code>-hookup_pin_sdo <i>pin</i></code>	Specifies the core-side hookup pin to be used for the scan data output signal during scan chain connection.

Command Reference for Encounter RTL Compiler

Design for Test

Note: When you specify this option, the RC-DFT engine does not validate the control ability of any logic between the top-level shift_enable signal and its designated hookup pin under test-mode setup.

`-max_length integer`

Specifies the maximum length that you allow for this scan chain.

If you omit this option, the maximum length defaults to the value of the `dft_max_length_of_scan_chains` design attribute.

Note: This option is ignored if the scan chain is defined with the `-complete` option, or if the number of flip-flop instances in a head, body, or tail segment exceeds the maximum value.

`-name name`

Specifies the name of the scan chain.

If you omit this option, a default name is used.

`-sdi sdi`

Specifies the scan data input signal.

- If you want to *create* a chain, specify a top-level port or a hierarchical pin name in case of an existing port or pin. If you want the tool to create the port, use the `-create_ports` option and a primary input port with the specified name will be created.
- If you want to *analyze* an existing chain, specify a top-level port, a hierarchical pin, subport, or a non-sequential instance pin.

`-sdo sdo`

Specifies the scan data output signal.

- If you want to *create* a chain, specify a top-level port or a hierarchical pin name in case of an existing port or pin. If you want the tool to create the port, use the `-create_ports` option and a primary output port with the specified name will be created.
- If you want to *analyze* an existing chain, specify a top-level port, a hierarchical pin, subport, or a non-sequential instance pin.

`-shared_select test_signal`

Specifies the select control signal to the mux inserted for a shared output port.

Command Reference for Encounter RTL Compiler

Design for Test

Default: The default shift-enable signal or chain-specific shift-enable signal is used as the select control signal to the mux.

`{-shared_output | -non_shared_output}`

Specifies whether an existing functional output port can be used as scan data output port. If the functional port can be used for scan data purposes, a mux is inserted in the scan data path by the `connect_scan_chains` command.

One of these options must be specified when the specified scan data output signal is already connected in the circuit.

`-shift_enable test_signal`

Designates a chain-specific shift-enable port or pin.

If you omit this option, the default shift-enable signal specified using a `define_dft shift_enable` constraint is used.

`-tail segment`

Indicates that the specified segment (an ordered set of scan flip-flops) must be placed at the tail (closest to the scan data output) of the scan chain. The segment must have been previously defined with a `define_dft xxx_segment` constraint, where `xxx` is either `abstract`, `fixed`, `floating`, `preserved`, or `shift_register`.

`-terminal_lockup {level_sensitive | edge_sensitive}`

Specifies the type of lockup element that configuration can insert at the tail end of the chain to connect to the specified scan data output signal.

If this option is not specified, no terminal lockup element will be inserted.

Note: This option only applies to the muxed scan style.

Examples

- The following example creates a chain containing 3 segments previously defined:

```
rc:/des*/test> define_dft scan_chain -sdi in[0] -sdo out[0] -shared_out \
    -head segHead -tail segTail -body segBody -name chain1
...
Info      : Added scan chain. [DFT-151]
           : scan chain successfully defined.
/designs/test/dft/scan_chains/chain1
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example analyzes an existing scan chain:

```
rc:/> define_dft scan_chain -name topChain -sdi SI -sdo SO -analyze  
...  
Info      : Added scan chain. [DFT-151]  
          : scan chain successfully defined.  
/designs/test/dft/report/actual_scan_chains/topChain
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Defining Scan Chains](#)
- [Creating Head, Body, and Tail Segments](#)
- [Analyzing Chains in a Scan-Connected Netlist](#)

[DFT-Related Commands](#) in *Interfacing between Encounter RTL Compiler and Encounter Conformal*

Affected by this constraint:

[define_dft abstract_segment](#) on page 628
[define_dft fixed_segment](#) on page 641
[define_dft floating_segment](#) on page 643
[define_dft preserved_segment](#) on page 671
[define_dft shift_enable](#) on page 686
[define_dft shift_register_segment](#) on page 689
[define_dft test_clock](#) on page 693

Affects these commands:

[connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788

Affected by this attribute:

[dft_max_length_of_scan_chains](#)
[dft_mix_clock_edges_in_scan_chains](#)
[dft_prefix](#)

Sets these attributes:

[Scan Chain Attributes](#)

define_dft scan_clock_a

```
define_dft scan_clock_a
  [-name name] [-no_ideal] driver
  [-period integer] [-divide_period integer]
  [-rise integer] [-divide_rise integer]
  [-fall integer] [-divide_fall integer]
  [ [-hookup_pin pin [-hookup_polarity string]]]
  [-configure_pad {tm_signal|se_signal}]
  | [-create_port]
```

Defines the scan clock of the master latch (`scan_clock_a`) of the clocked LSSD scan cell. The `scan_clock_a` signal controls the scan shifting of the master latch and is required for the clocked-LSSD scan style. The signal is created with active high polarity.

You can define only one signal for the design. If you specify more than one signal, the last definition overwrites the existing one.

The command returns the directory path to the `test_signal` object that it creates. You can find the object created by the `define_dft scan_clock_a` constraints in:

```
/designs/design/dft/test_signals
```

Options and Arguments

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the `scan_clock_a` signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port` Specifies whether to create the port if it does not exist.

`-divide_fall integer`

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 100

Command Reference for Encounter RTL Compiler

Design for Test

`-divide_period integer`

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 1

`-divide_rise integer`

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 100

`driver`

Specifies the driving pin or port for the scan clock of the master latch (`scan_clock_a`) of the clocked LSSD scan cell.

`-fall integer`

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 60

`-hookup_pin pin`

Specifies the core-side hookup pin to be used for the `scan_clock_a` signal during scan chain connection.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level `scan_clock_a` signal and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted|non_inverted}`

Specifies the polarity of the `scan_clock_a` signal at the core-side hookup pin.

`-name name`

Specifies the `test_signal` object name of the `scan_clock_a` signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

`-no_ideal`

Marks the `scan_clock_a` signal as non-ideal. This allows buffering of the `scan_clock_a` network during optimization.

By default, the `scan_clock_a` signal is marked ideal.

Command Reference for Encounter RTL Compiler

Design for Test

Note: If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the `scan_clock_a` signal

`-period integer` Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 50000 (20 MHz test clock)

`-rise integer` Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 50

Example

- The following example defines `sca` as the driver for the `scan_clock_a` signal and assigns `SCA` as the `test_signal` object name.

```
define_dft scan_clock_a -name SCA sca
```

Related Information

[Defining LSSD Scan Clock Signals in Design for Test in Encounter RTL Compiler](#)

Affects these commands: [connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788
[write_atpg](#) on page 805

Sets these attributes: [Test Signal Attributes](#)

Affected by this attribute: [dft_scan_style](#)

define_dft scan_clock_b

```
define_dft scan_clock_b
  [-name name] [-no_ideal] driver
  [-period integer] [-divide_period integer]
  [-rise integer] [-divide_rise integer]
  [-fall integer] [-divide_fall integer]
  [ [-hookup_pin pin [-hookup_polarity string]]]
  [-configure_pad {tm_signal|se_signal}]
  | [-create_port]
```

Defines the scan clock of the slave latch (`scan_clock_b`) of the clocked LSSD scan cell. The `scan_clock_b` signal controls the scan shifting of the slave latch and is required for the clocked-LSSD scan style. The signal is created with active high polarity.

You can define only one signal for the design. If you specify more than one signal, the last definition overwrites the existing one.

The command returns the directory path to the `test_signal` object that it creates. You can find the object created by the `define_dft scan_clock_b` constraints in:

```
/designs/design/dft/test_signals
```

Options and Arguments

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the `scan_clock_b` signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port` Specifies whether to create the port if it does not exist.

`-divide_fall integer`

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 100

Command Reference for Encounter RTL Compiler

Design for Test

`-divide_period integer`

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 1

`-divide_rise integer`

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 100

`driver`

Specifies the driving pin or port for the scan clock of the slave latch (`scan_clock_b`) of the clocked LSSD scan cell.

`-fall integer`

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 80

`-hookup_pin pin`

Specifies the core-side hookup pin to be used for the `scan_clock_b` signal during scan chain connection.

Note: When you specify this option, the RC-DFT engine does not validate the controlability of any logic between the top-level `scan_clock_b` signal and its designated hookup pin under test-mode setup.

`-hookup_polarity {inverted|non_inverted}`

Specifies the polarity of the `scan_clock_b` signal at the core-side hookup pin.

`-name name`

Specifies the `test_signal` object name of the `scan_clock_b` signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

`-no_ideal`

Marks the `scan_clock_b` signal as non-ideal. This allows buffering of the `scan_clock_b` network during optimization.

By default, the `scan_clock_b` signal is marked ideal.

Command Reference for Encounter RTL Compiler

Design for Test

Note: If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the `scan_clock_b` signal

`-period integer` Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 50000 (20 MHz test clock)

`-rise integer` Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 70

Example

- The following example defines `sca` as the driver for the `scan_clock_b` signal and assigns `SCB` as the `test_signal` object name.

```
define_dft scan_clock_b -name SCB sca
```

Related Information

[Defining LSSD Scan Clock Signals in Design for Test in Encounter RTL Compiler](#)

Affects these commands: [connect_scan_chains](#) on page 620
[report_dft_chains](#) on page 788
[write_atpg](#) on page 805

Sets these attributes: [Test Signal Attributes](#)

Affected by this attribute: [dft_scan_style](#)

define_dft shift_enable

```
define_dft shift_enable [-name name] -active {low|high}
    [-default] [-no_ideal]
    [ [-hookup_pin pin [-hookup_polarity string]]
        [-configure_pad {tm_signal|se_signal}]
    | -create_port ]
    {pin|port} [-design design]
```

Specifies the name and active value of the input signal that activates scan shifting. The input signal can be defined on a top-level port or an internal driving pin. This type of input signal is required by the `muxed_scan` style. The active value of the shift-enable signals is propagated through the design by the `check_dft_rules` command.

The command returns the directory path to the `test_signal` object that it creates. You can find the objects created by the `define_dft shift_enable` constraints in:

```
/designs/design/dft/test_signals
```

Options and Arguments

`-active {low | high}`

Specifies the active value for the shift-enable signal.

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the shift-enable signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft test_mode` or `define_dft shift_enable` constraint.

`-create_port`

Specifies whether to create the port if it does not exist.

`-default`

Designates the specified signal as the default shift-enable signal for chains for which you omit the `-shift-enable` option.

Note: You can designate only one signal as the default shift-enable signal.

`-design design`

Specifies the name of the design for which the shift enable is defined.

Command Reference for Encounter RTL Compiler

Design for Test

If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used.

-hookup_pin *pin* Specifies the core-side hookup pin to be used for the top-level shift-enable signal during DFT synthesis.

Note: When you specify this option, the RC-DFT engine does not validate the controllability of any logic between the top-level shift_enable signal and its designated hookup pin under test-mode setup.

-hookup_polarity {inverted | non_inverted}

Specifies the polarity of the shift-enable signal at the core-side hookup pin.

-name *name* Specifies the `test_signal` object name of the shift-enable signal.

If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.

-no_ideal Marks the shift-enable signal as non-ideal. This allows buffering of the shift-enable network during optimization.

Default: The shift-enable signal is marked ideal.

Note: If the test signal is marked as ideal, RTL Compiler sets the `ideal_network` attribute to `true` on the pin or port for the shift-enable signal

{*pin|port*} Specifies the driving pin or port for the shift-enable signal.

Note: If multiple designs are loaded and you did not specify the `-design` option, you can also specify the full path to the driver.

Example

- When the following constraint is given, the `check_dft_rules` command propagates a logic1 from the `p_top/SE` pin into the design.

```
define_dft shift_enable -active low -hookup_pin p_top/SE -hookup_polarity inverted
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Defining Shift-Enable Signals in Design for Test in Encounter RTL Compiler](#)

Affects these commands: [check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

[insert_dft_shadow_logic](#) on page 763

[report_dft_chains](#) on page 788

Sets these attributes: [Test Signal Attributes](#)

define_dft shift_register_segment

```
define_dft shift_register_segment [-name segment_name]  
        -start_flop instance  
        -end_flop instance
```

Defines a shift register. Because a shift register is a shiftable scan chain segment, the RC-DFT engine can use the functional path of the shift register as the scan path by only scan-replacing the first flop in the shift register segment, while maintaining the existing connectivity of the flops.

Note: A shift register segment can only contain flops driven by the same clock and same clock edge.

A shift register is a user-specified scan segment which can be associated with either

- A user-defined top-level chain—created using the [define_dft scan_chain](#) command
- A tool-created scan chain—created using the [connect_scan_chains](#) command

The command returns the directory path to the object that it creates. You can find the objects created by the `define_dft shift_register_segment` constraint in:

```
/designs/top_design/dft/scan_segments
```

Note: Shift register segments are only supported for the muxed scan style.

Options and Arguments

- end_flop *instance* Specifies the last flop in the shift register. Specify the hierarchical instance name of the flop.
- name *segment_name* Defines a name for the segment that you can use to reference in the [define_dft scan_chain](#) constraint.
- start_flop *instance* Specifies the first flop in the shift register. Specify the hierarchical instance name of the flop.

Example

- The following example defines a shift register.

```
define_dft shift_register_segment -name myreg \  
        -start_flop *seq/out_reg_0 -end_flop *seq/out_reg_7
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Manually Identifying Shift Registers](#)
- [Creating Head, Body, and Tail Segments](#)
- [Identifying Shift Registers in a Mapped Netlist before Creating the Scan Chains](#)

Affects this constraint: [define_dft_scan_chain](#) on page 674

Affects these commands: [connect_scan_chains](#) on page 620

[report_dft_chains](#) on page 788

Related command: [identify_shift_register_scan_segments](#) on page 710

Sets these attributes: [Scan Segment Attributes](#)

define_dft tap_port

```
define_dft tap_port {pin|port} [-create_port]
    -type {tck | tdi | tdo | tdo_enable | tms | trst}
    [-hookup_pin {pin|port}] [-hookup_polarity string]
    [design]
```

Defines a TAP port (JTAG signal). Specifies the internal hookup pin for a JTAG signal. These hookup pins will be used when inserting boundary scan logic, JTAG macro, MBIST logic and PTAM logic, to make connections from its DFT logic to the hookup pins specified for each of the JTAG signals.

Note: You can only specify this command before inserting the boundary scan logic or the JTAG macro. When defining an existing JTAG macro using `define_dft jtag_macro`, the tap ports must be defined with the `define_dft tap_port` command.

You can find the objects created by the `define_dft tap_port` constraints in:

```
/designs/design/dft/boundary_scan/tap_ports
```

Options and Arguments

<code>-create_port</code>	Creates the port if it does not exist.
<code>design</code>	Specifies the design for which the tap ports are being defined.
<code>-hookup_pin {pin port}</code>	Specifies the core-side hookup pin to be used for the top-level JTAG signal during DFT synthesis.
<code>-hookup_polarity {non_inverted inverted}</code>	Specifies the polarity of the JTAG signal at the core-side hookup pin. <i>Default:</i> non_inverted
<code>{pin port}</code>	Specifies the driving pin or port for the JTAG signal.
<code>-type {tck tdi tdo tdo_enable tms trst}</code>	Specifies the type of TAP port being created.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Hookup Requirements for TAP Ports](#)
- [JTAG-Controlled Scan Modes](#)

Sets these attributes: [TAP Port Attributes](#)

define_dft test_clock

```
define_dft test_clock -name test_clock
  [-design design] [-domain test_clock_domain]
  [-period integer] [-divide_period integer]
  [-rise integer] [-divide_rise integer]
  [-fall integer] [-divide_fall integer]
  [-hookup_pin pin [-hookup_polarity string]]
  [-controllable] {pin|port} [{pin|port}] ...
```

Defines a test clock and associates a test clock waveform with the clock. The test clock waveform can be different from the system clocks.

If you do not define test clocks, the DFT rule checker automatically analyzes the test clocks and creates these objects with a default waveform. The waveform information is useful in determining how to order scan flip-flops in a chain, and where to insert data-lockup elements in the chain.

Test clock waveforms are used to order flip-flops that belong to the same DFT test clock domain to minimize the addition of lockup elements. Flip-flops that are triggered first are ordered and connected last in a chain.

The command returns the directory path to the `test_clock` object that it creates. You can find the objects created by the `define_dft test_clock` constraints in:

```
/designs/design/dft/test_clock_domains
```

Options and Arguments

`-controllable`

When specifying an internal pin for a test clock, this option indicates that the internal clock pin is controllable in test mode (for example, Built-in-Self-Test (BIST)). If you do not specify this option, the rule checker must be able to trace back from the internal pin to a controllable top-level clock pin.

Note: If you specify an internal pin as being controllable, you need to ensure that this pin can be controlled for the duration of the test cycle. The tool will *not* validate your assumption.

`-design design`

Specifies the name of the design for which the test clock is defined.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

Command Reference for Encounter RTL Compiler

Design for Test

`-divide_fall integer`

Together with the `-fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 100

`-divide_period integer`

Together with the `-period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 1

`-divide_rise integer`

Together with the `-rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 100

`-domain test_clock_domain`

Specifies the DFT clock domain associated with the test clock.

Clocks belonging to the same domain can be mixed in a chain.

If you omit this option, a new DFT clock domain is created and associated with the test clock.

Flip-flops belonging to different test clocks in the same domain can be mixed in a chain. Lockup elements can be added between the flip-flops belonging to different test clocks.

`-fall integer`

Together with the `-divide_fall` option, determines the time that the falling edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-fall` by `-divide_fall`.

Default: 90

`-hookup_pin pin`

Specifies the core-side hookup pin to be used for the top-level test clock during DFT synthesis.

Note: When you specify this option, the RC-DFT engine does not validate the controllability of any logic between the top-level hookup pin under test-mode setup.

`-hookup_polarity {inverted | non_inverted}`

Specifies the polarity of the test clock signal at the core-side hookup pin.

`-name test_clock`

Specifies the name of the test clock that is being defined.

Each clock object in your design must have a unique name. If you define a new test clock with the same name as an existing clock, an error message will be issued.

Note: The clock name allows you to search for the clock later (through the `find` command) or to recognize it in reports.

`-period integer`

Together with the `-divide_period` option, determines the clock period interval. The clock period is specified in picoseconds and is derived by dividing `-period` by `-divide_period`.

Default: 50000 (20 MHz test clock)

`{pin|port}`

Specifies the test clock input pin or port.

If you specify multiple pins, these pins are assumed to have zero skew: they can be mixed without lockup latches.

`-rise integer`

Together with the `-divide_rise` option, determines the time that the rising edge occurs with respect to the beginning of the clock period. The time is specified as a percentage of the period and is derived by dividing `-rise` by `-divide_rise`.

Default: 50

Example

The following example defines three test clocks, four test clock ports and two DFT clock domains.

```
define_dft test_clock -name CLK1X -domain domain_1 -period 20000 CLK1
define_dft test_clock -name CLK2X -domain domain_1 -period 20000 CLK2 CLK2b
define_dft test_clock -name CLK3X -domain domain_2 -period 20000 CLK3
```

The four test clock ports are CLK1, CLK2, CLK2b, and CLK3. Test clock CLK2X comprises equivalent test clocks CLK2 and CLK2B (they can be mixed in the same scan chain without any lockup element). Test clocks CLK1X, CLK2X belong to the same DFT clock domain and

are compatible (requires lockup elements between compatible chain segments triggered by the different test clocks). Test clock CLK3X belongs to its own domain.

Related Information

See the following section in *Design for Test in Encounter RTL Compiler*:

- [Defining Test Clock Waveforms](#)
- [Defining Internal Clock Branches as Separate Test Clocks](#)
- [Defining Equivalent Test Clocks for Different Top-level Clock Pins](#)
- [Defining an Internal Pin as Test Clock](#)

Affects these commands:

[check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

[fix_dft_violations](#) on page 703

[report_dft_chains](#) on page 788

Sets these attributes:

[Test Clock Attributes](#)

define_dft test_mode

```
define_dft test_mode [-name name] -active {low | high}
  [-no_ideal] [-scan_shift]
  [ [-hookup_pin pin [-hookup_polarity string]]
    [-configure_pad {tm_signal|se_signal}]
  | [-create_port | -shared_in | -test_only] ]
  {pin|port} [-design design]
```

Specifies the input signal and constant value that is assigned during a test session. The input signal can be defined on a top-level port or an internal driving pin.

The active value of the test mode signals is propagated through the design by the `check_dft_rules` command. Unless defined with the `-scan_shift` option, the test signal is expected to stay active throughout a test session.

The command returns the directory path to the `test_signal` object that it creates. You can find the objects created by the `define_dft test_mode` constraints in:

```
/designs/design/dft/test_signals
```

Options and Arguments

`-active {low | high}`

Specifies the active value for the test mode signal.

`-configure_pad {tm_signal | se_signal}`

Specifies the test signal that the RC-DFT engine must use if it needs to configure the pad connected to the test mode signal to control the data direction during test mode.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-create_port`

Specifies whether to create the port if it does not exist.

Note: This option cannot be specified with the `-shared_in` option.

`-design design`

Specifies the name of the design for which the test mode signal is defined.

Note: If you omit the design name and multiple designs are loaded, the top-level design of the current directory of the design hierarchy is used. You can also specify the full path to the driver.

Command Reference for Encounter RTL Compiler

Design for Test

-hookup_pin <i>pin</i>	Specifies the core-side hookup pin to be used for the top-level test-mode signal during DFT synthesis. Note: When you specify this option, the RC-DFT engine does not validate the controllability of any logic between the top-level test-mode signal and its designated hookup pin under test-mode setup.
-hookup_polarity {inverted non_inverted}	Specifies the polarity of the test-mode signal at the core-side hookup pin.
-name <i>name</i>	Specifies the <code>test_signal</code> object name of the test signal. If you omit this option, the RC-DFT engine assigns a name based on the hierarchical path of the driver, using underscores as delimiters in the path.
-no_ideal	Marks the test-mode signal as non-ideal. This allows buffering of the test-mode network during optimization. <i>Default:</i> The test-mode signal is marked ideal. Note: If the test signal is marked as ideal, RTL Compiler sets the <code>ideal_network</code> attribute to <code>true</code> on the pin or port for the test signal.
{ <i>pin port</i> }	Specifies the driving pin or port for the test signal. Note: If multiple designs are loaded and you did not specify the <code>-design</code> option, you can also specify the full path to the driver.
-scan_shift	Indicates that this test signal should only be held to its test-mode active value during the scan shift operation of the tester cycle. This option is used to designate those test signals which must be held to their non-controlling functional values to prevent the state of the flip-flops from being asynchronous set or reset while ATPG data is being shifted into or out of the scan chains. As a consequence of specifying this option, the test signal will be treated as a non-scan clock signal by the ATPG tool. This means ATPG can pulse the pin during the capture window, but will leave it in the off state during scan shifting.

If this option is not specified, the test signal will be held to its test-mode active value for the duration of the tester cycle.

 *Important*

Specify this option for the appropriate test signals (such as asynchronous set and reset signals and similar signals) to ensure that these test signals will not get constrained in the `write_do_lec` dofile. Not specifying this option for the appropriate test signals will result in over constraining the `write_do_lec` dofile which can lead to false EQs.

`-shared_in`

Specifies whether the input port is also used as a functional port.

By default, the signal applied to the specified driving pin or port is considered to be a dedicated test signal.

Note: This option cannot be specified with the `-create_port` option.

 *Important*

Specify this option for the shared test signals (such as those driving functional logic in the golden design) to ensure that these test signals will not get constrained in the `write_do_lec` dofile. Not specifying this option for a shared test signal will result in over constraining the `write_do_lec` dofile which can lead to false EQs.

`-test_only`

Specifies that the input port will only be used for test purposes.

When this option is specified, the tool might add a constraint for the port in the do file generated by the `write_do_lec` command.

Example

- When the following constraint is given, the `check_dft_rules` command propagates a logic1 from the `pad_top/TM` pin into the design.

```
define_dft test_mode -active high -hookup_pin pad_top/TM  
-hookup_polarity non_inverted
```

Related Information

[Defining Test Mode Signals in Design for Test in Encounter RTL Compiler](#)

[DFT-Related Commands in Interfacing between Encounter RTL Compiler and Encounter Conformal](#)

Affects these commands: [check_dft_rules](#) on page 589

[fix_dft_violations](#) on page 703

[insert_dft_shadow_logic](#) on page 763

[insert_dft_test_point](#) on page 768

Sets these attributes: [Test Signal Attributes](#)

dft_trace_back

```
dft_trace_back  
  [-mode integer] [-polarity]  
  [-continue] [-print]  
  {port|pin}
```

Returns the pin or port found by tracing back one level from the specified pin or port based on the requested mode. If a constant is encountered, the command returns 0 or 1.

Options and Arguments

<code>-mode <i>integer</i></code>	Specifies the mode for tracing back. <ul style="list-style-type: none">■ 0 does not perform constant propagation■ 1 performs tied-constant propagation■ 2 performs tied-constant and test-mode propagation■ 3 performs tied-constant, test-mode and shift-enable propagation <p><i>Default:</i> 3</p>
<code>-continue</code>	Specifies to continue the trace back until a primary input, complex gate, or sequential gate is reached. Additionally, the trace will terminate if a logic constant is returned for the trace back pin.
<code>{pin port}</code>	Specifies the pin or port from which to start the trace back.
<code>-polarity</code>	Specifies whether to report if the polarity changed through the trace. A returned value of 0 indicates no inversion. A returned value of 1 indicates an inversion.
<code>-print</code>	Prints the pin and polarity at every trace back.

Examples

- Following command traces back without performing constant propagation.

```
rc:/> dft_trace_back -mode 0 /designs/top/instances_hier/g121/pins_in/CME  
/designs/Top/ports_in/cme
```

Command Reference for Encounter RTL Compiler

Design for Test

- Following command traces back using the default mode. In this case a constant logic 1 value is reported.

```
rc:/> dft_trace_back /designs/top/instances_hier/g121/pins_in/CME  
1
```

- Following command traces back using the default mode and requests to report whether there was a change in polarity. In this case, a constant logic 1 with no change in logic polarity is reported.

```
rc:/> dft_trace_back /designs/top/instances_hier/g121/pins_in/CME -polarity  
1 0
```

fix_dft_violations

```
fix_dft_violations
  { -clock -test_control test_signal
    [-test_clock_pin {pin|port} [-rise | -fall]]
  | {-async_set | -async_reset | -async_set -async_reset}
    -test_control test_signal [-async_control test_signal]
    [-insert_observe_scan
      -test_clock_pin {pin|port} [-rise | -fall]]}
  [-violations violation_object_id_list]
  [-tristate_net]
  [-xsource [-exclude_xsource instance...]]
  [-preview] [-dont_check_dft_rules] [-dont_map]
  [-design design]
```

Automatically fixes either

- All DFT violations of the specified types (clock, asynchronous set, asynchronous reset, tristate, or xsource).

Only the specified violation types are fixed. If you allow to fix asynchronous set and reset violations using the same test mode signal, you can request both types to be fixed at the same time.

- The *identified* violations

You can further limit the violations that RTL Compiler must fix to by specifying the violation ID (through the `-violations` option).

Note: Currently, clock violations are only fixed for the muxed scan style.

Options and Arguments

`-async_control test_signal`

Specifies the name of the test signal to use to control the asynchronous violations to be fixed.

`-async_reset` Fixes the asynchronous reset violations on all instances.

`-async_set` Fixes the asynchronous set violations on all instances.

`-clock` Fixes the clock violations on all instances.

`-design design` Specifies the name of the design whose violations must be checked.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

Command Reference for Encounter RTL Compiler

Design for Test

`-dont_check_dft_rules`

Prevents the DFT rules from being checked automatically after fixing violations.

`-dont_map`

Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

`-exclude_xsource instance`

Specifies instances to exclude from automatic fixing of X-source violations.

`-insert_observe_scan`

Inserts a flip-flop for observability. If you fix DFT violations in a generic netlist, the flip-flop is mapped to a scan flip-flop during synthesis. If you fix DFT violations in a mapped netlist, the flip-flop is mapped to a scan flip-flop. You need to rerun the `check_dft_rules` command to update the DFT status of the observability flop prior to connecting it in a scan chain.

Note: This option can only be used when fixing an asynchronous set reset violation and requires the `-test_clock_pin` option.

`{pin | port}`

Specifies the test signal pin or port to use to control the set or reset. By default, the set or reset are controlled by the `-test_control` option.

To specify this option, the pin or port must first be declared as a `test_mode` signal using the `define_dft test_mode` constraint.

`-preview`

Reports how the violations will be fixed, without making modifications to the netlist.

`[-rise | -fall]`

Specifies to use the rising or falling edge of the test clock to fix the DFT violation during test-mode operation.

Default: `-rise`

`-test_clock_pin {pin | port}`

Specifies the test clock signal to be used. Specify the pin or port from where the clock signal originates. In most applications, the clock pin or port is identified when checking the DFT rules.

This option is optional when fixing clock violations. By default, the RC-DFT engine performs a clock trace to identify a controllable test clock that appears in the fanin cone of the clock violation and uses this test clock to fix the actual clock violation.

This option is required when you want to insert observability flip-flops when fixing async violations. In this case, the test clock signal drives the clock pin of the observation flip-flops during test-mode operation.

`-test_control test_signal`

Specifies the name of the test signal to use to fix the violation.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

`-tristate_net` Specifies to fix tristate net contention design rules violations.

`-violations violation_object_id_list`

Fixes the violations that are identified by their object name.

Note: The `check_dft_rules` command creates a `violations` directory in the design hierarchy under `/designs/design/dft/report`. The objects in this directory correspond to the violations found during the last execution of the `check_dft_rules` command. Use the `report dft_violations` command to list all remaining violations in the design.

`-xsource`

Specifies to fix X-Source design rules violations.

Examples

- The following example instructs RTL Compiler to fix all clock, async set, and asynch reset violations using test mode signal `tm` and test clock `CK1` using the rising edge as the active edge.

```
fix_dftViolations -clock -async_set -async_reset  
-test_control tm -insert_observe_scan -test_clock_pin CK1 -rise
```

If you do not want to use the same test clock to fix the clock violations and to drive the clock pin of the observation flip-flops, you need to enter two commands. For example,

```
fix_dftViolations -clock -test_control tm  
fix_dftViolations -async_set -async_reset -test_control tm  
-insert_observe_scan -test_clock_pin CK1 -rise
```

Command Reference for Encounter RTL Compiler

Design for Test

In this case, the RC-DFT engine automatically determines which test clock to use to fix the clock violations.

- The following example instructs RTL Compiler to fix all clock, async set, and asynch reset violations using test mode signal `tm` and test clock `CK1` using the rising edge as the active edge.

```
fix_dft_violations -clock -async_set -async_reset  
-test_control tm -test_clock_pin CK1 -rise
```

- The following example instructs RTL Compiler to fix violations `vid_1` and `vid_3` if they are violations of type `async_set`.

```
fix_dft_violations -violations {vid_1 vid_3} -async_set -test_control tm
```

Related Information

[Fixing DFT Rule Violations in Design for Test in Encounter RTL Compiler](#)

Affected by these constraints: [define_dft shift_enable](#) on page 686

[define_dft test_clock](#) on page 693

[define_dft test_mode](#) on page 697

Affects these commands: [check_dft_rules](#) on page 589

[report dft_registers](#) on page 790

[report dft_violations](#) on page 792

[synthesize](#) on page 348

Sets these attributes: [dft_status](#)

[dftViolation](#)

[Violations Attributes](#)

fix_scan_path_inversions

`fix_scan_path_inversions actual_scan_chain...`

Fixes inversions for every scan element in the scan path. The command inserts inverters as required in a scan chain to reset the flops in the chain to `logic_0`.

Options and Arguments

`actual_scan_chain`

Specifies the scan chain(s) to undergo analysis and to insert inverters.

Related Information

[Fixing Scan Path Inversions in Design for Test in Encounter RTL Compiler](#)

identify_multibit_cell_abstract_scan_segments

```
identify_multibit_cell_abstract_scan_segments  
  [-dont_check_dft_rules] [-preview]  
  [-libcell libcell...] [-design design]
```

Identifies multi-bit scan cells in the design, and defines scan abstract segments for each instance of the multi-bit scan cell.

Multi-bit scan cells implemented using either a parallel or serial bit approach are supported by the tool.

Options and Arguments

<code>-design design</code>	Specifies the name of the top-level design on which to identify abstract segments for multi-bit scan cells. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
<code>-dont_check_dft_rules</code>	Prevents the DFT rules from being automatically checked after identifying abstract segments for multi-bit scan cells.
<code>-libcell libcell</code>	Specifies the multi-bit library cells on which to perform abstract segment identification.
<code>-preview</code>	Reports the identified abstract scan segments without defining them.

Examples

- The following example identifies a parallel multi-bit scan cell with two bits; where each bit would be defined as an abstract scan segment:

```
rc:/> identify_multibit_cell_abstract_scan_segments -preview
```

Would execute command:

```
define_dft abstract_segment -length 1 -sdi SI1 -sdo Q1  
-shift_enable_port SE1 -active high -clock_port CP -rise  
-libcell/libraries/tcbn65ulp_c070701wc2/libcells/DUALSDFQD0 -name DUALSDFQD0
```

Would execute command:

```
define_dft abstract_segment -length 1 -sdi SI2 -sdo Q2  
-shift_enable_port SE2 -active high -clock_port CP -rise  
-libcell /libraries/tcbn65ulp_c070701wc2/libcells/DUALSDFQD0 -name DUALSDFQD0
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example identifies a serial multi-bit scan cell of length 4; where each instance of the cell would be defined as an abstract scan segment:

```
rc:/> identify_multibit_cell_abstract_scan_segments -preview
```

Would execute command:

```
define_dft abstract_segment -length 4 -sdi SI -sdo Q4  
-shift_enable_port SM -active high -clock_port CK -rise  
-libcell /libraries/cs60ale_uc_scan/libcells/YSDM4ALU1 -name YSDM4ALU1
```

Related Information

[Mapping to Multi-Bit Scan Cells in Design for Test in Encounter RTL Compiler](#)

identify_shift_register_scan_segments

```
identify_shift_register_scan_segments
  [-min_length integer] [-max_length integer]
  [-preview] [-incremental]
```

Identifies all shift registers in the design whose minimum length either satisfies the default minimum length, or the specified minimum and maximum length values.

The RC-DFT engine uses the following naming convention for automatically identified shift-register segments:

DFT_AutoSegment_n

You can find the automatically identified shift-register segments in:

/designs/top_design/dft/scan_segments

Note: A shift register segment contains flops driven by the same clock and same clock edge.

A shift register is a scan segment which can be associated with either

- A user-defined top-level chain—created using the [define_dft_scan_chain](#) command
- A tool-created scan chain—created using the [connect_scan_chains](#) command

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-incremental Adds new shift register segments in incremental mode, without changing already identified shift register segments stored in /designs/design/dft/scan_segments

Note: Without this option, the existing identified shift register segments will be removed and new segments will be identified based on the new values specified for the command options.

-max_length integer

Specifies the maximum length that an automatically identified shift register can have.

If the length of a functional shift register exceeds this length, it will be broken in multiple scan segments.

Command Reference for Encounter RTL Compiler

Design for Test

Note: There is no default for the maximum length.

`-min_length integer`

Specifies the minimum length a shift register must have to be automatically identified by the tool.

Default: 2

`-preview`

Reports which shift register segments will be identified, without adding them to the list of scan segments.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Automatically Identifying Shift Registers](#)
- [Identifying Shift Registers in a Mapped Netlist before Creating the Scan Chains](#)
- [Analyzing Chains in a Scan-Connected Netlist](#)

Sets this attribute:

[user_defined_segment](#)

identify_test_mode_registers

```
identify_test_mode_registers
  { -fixed_value_register_file file
  | {-stil file [-macro string] | -mode_init file}
    [-library string] [-et_log string]
    [-continue_with_severe_warnings] }
  [-design design] [-preview]
```

Identifies all internal registers whose output signals must remain constant during test mode and generates the corresponding test-mode signals required for the RC-DFT engine.

The fixed-value registers can be identified

- From an existing ET log file
- By invoking Encounter Test and simulating a mode initialization sequence

Note: In the latter case, you need to have the Encounter Test software installed and your operating system PATH environment variable must include the path to the Encounter Test software. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-continue_with_severe_warnings

Continues the Encounter Test run even when severe warnings are issued.

-design design

Specifies the name of the design for which to define the test-mode signals.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-et_log string

Specifies the log file for Encounter Test.

-fixed_value_register_file file

Specifies an ET log file from a previous run that contains a list of fixed value registers.

When this option is specified, RTL Compiler can parse this file instead of invoking Encounter Test to get the list of fixed value registers.

Command Reference for Encounter RTL Compiler

Design for Test

	<p>Note: This option cannot be specified with the <code>-stil</code>, <code>-mode_init</code>, <code>-macro</code>, <code>-library</code>, <code>-et_log</code> and <code>-continue_with_severe_warnings</code> options.</p>
<code>-library string</code>	Specifies the list of Verilog structural library files. Specify the list in a quoted string. Refer to the <code>write_et_atpg -library</code> option description for additional information.
	<p>Note: This option is only required when you invoke this command on a mapped netlist.</p>
<code>-macro string</code>	Specifies the name of the macro in the STIL file that contains the initialization vectors to be simulated.
	<p><i>Default:</i> <code>test_setup</code></p>
<code>-mode_init file</code>	Specifies the name of the file that contains the mode initialization sequence in TBDpatt format.
<code>-preview</code>	Reports the fixed value registers but does not set the test mode values.
<code>-stil file</code>	Specifies the name of the STIL file that contains the mode initialization sequence.

Examples

- The following command requests a report of the list of the registers with fixed values.

```
rc:/> identify_test_mode_registers -stil newStil -prev
Identifying internal registers with fixed value outputs under test_mode setup.
... Creating intermediate files

WARNING : No user defined shift enable signal found.
ATPG interface file may contain incomplete information
Cadence Design Systems RC file: Cadence ATPG file created successfully.

-----
... Identifying the fixed value registers
-----

Test Function      Block Name
+TI               tc/ts_reg[0]
+TI               tc/ts_reg[1]
+TI               tc/ts_reg[2]
-----
Note: The +/-TI Test Function flag denotes the active logic value that a signal
is to be held to during test mode.
  +TI denotes a logic value 1; -TI denotes a logic value 0
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following command creates the test signals and automatically reruns the DFT rule checker.

```
rc:/> identify test_mode_registers -stil newStil
Identifying internal registers with fixed value outputs under test_mode setup.
... Creating intermediate files

WARNING : No user defined shift enable signal found.
ATPG interface file may contain incomplete information
Cadence Design Systems RC file: Cadence ATPG file created successfully.

-----
... Identifying the fixed value registers
-----
INFO: Setting active high test mode signal on tc/ts_reg[0]/q
INFO: Setting active high test mode signal on tc/ts_reg[1]/q
INFO: Setting active high test mode signal on tc/ts_reg[2]/q
-----
Checking DFT rules for 'top' module under 'muxed_scan' style

...
rc:/> ls dft/test_signals
/designs/top/dft/test_signals:
./                                     rst                         tc_ts_reg[1]_q
incr                                tc_ts_reg[0]_q                         tc_ts_reg[2]_q
```

Related Information

[Identifying Fixed-Value Registers in Design for Test in Encounter RTL Compiler](#)

Affected by these constraints: [define_dft_test_mode](#) on page 697

[define_dft_test_clock](#) on page 693

Sets this attribute: [user_defined_signal](#)

insert_dft

```
insert_dft
  { boundary_scan | compression_logic
  | dfa_test_points | jtag_macro
  | lockup_element | logic_bist | mbist | opcg
  | ptam | rrfa_test_points | scan_power_gating
  | shadow_logic | test_point | user_test_point
  | wrapper_cell | wrapper_mode_decode_block}
```

Inserts DFT test logic.

Options and Arguments

<code>boundary_scan</code>	Inserts boundary scan cells and the corresponding JTAG controller.
<code>compression_logic</code>	Inserts compression logic in a design that requires a fixed number of scan compression channels.
<code>dfa_test_points</code>	Inserts test points based on Deterministic Fault Analysis.
<code>jtag_macro</code>	Inserts a JTAG Macro controller into a netlist.
<code>lockup_element</code>	Inserts lockup elements in the specified analyzed scan chains.
<code>logic_bist</code>	Inserts Logic Bist (LBist) logic into the design.
<code>mbist</code>	Inserts Memory Built-In-Self-Test (MBIST) logic to test targeted memories in the design.
<code>opcg</code>	Inserts OPCG logic that generates on-chip launch and capture clocks for testing of at-speed delay defects.
<code>ptam</code>	Inserts Power Test Access Mechanism (PTAM) control logic into the design.
<code>rrfa_test_points</code>	Inserts test points based on
<code>scan_power_gating</code>	Inserts gating logic at selected flop outputs to minimize switching power during scan shift
<code>shadow_logic</code>	Inserts DFT shadow logic to enable testing of shadow logic around a module.
<code>test_point</code>	Inserts a native test point.
<code>user_test_point</code>	Inserts a user-defined test point.
<code>wrapper_cell</code>	Inserts an IEEE-1500 style core-wrapper cell.

Command Reference for Encounter RTL Compiler

Design for Test

`wrapper_mode_decode_block`

Builds a 1500 mode decode block based on the scan modes that were defined with type `wrapper`.

Related Information

Related commands:

[insert_dft boundary_scan](#) on page 717
[insert_dft dfa_test_points](#) on page 732
[insert_dft jtag_macro](#) on page 736
[insert_dft lockup_element](#) on page 740
[insert_dft logic_bist](#) on page 741
[insert_dft mbist](#) on page 743
[insert_dft opcg](#) on page 749
[insert_dft ptam](#) on page 751
[insert_dft rrfa_test_points](#) on page 754
[insert_dft scan_power_gating](#) on page 760
[insert_dft shadow_logic](#) on page 763
[insert_dft test_point](#) on page 768
[insert_dft user_test_point](#) on page 773
[insert_dft wrapper_cell](#) on page 775
[insert_dft wrapper_mode_decode_block](#) on page 780

insert_dft boundary_scan

```
insert_dft boundary_scan [-design design]
    [-comp_enables_high port [port]...]
    [-comp_enables_low port [port]...]
    [-exclude_ports port [port]...]
    [-functional_clocks port [port]...]
    [-custom_cell_directory string]
    [-bcells_location instance]
    [-jtag_macro_location instance]
    [-pinmap_file file | -physical]
    [-power_on_reset pin|port]
    [-tck port] [-tdi port] [-tdo port]
    [-tms port] [-trst port]
    [-dont_map] [-preview]
    [-preserve_tdo_connection]
```

Inserts boundary scan cells and the corresponding JTAG Macro (if it is not yet instantiated in the design).

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-bcells_location instance

Specifies the instance in which to insert the boundary scan cells. By default, the boundary scan cells are inserted in the same hierarchy as their respective pad cells in the design.

-comp_enables_high (-comp_enables_low) port...

Specifies that the compliance value for the specified ports is active high (low) during functional mode. The compliance enable value is the value that a test port (test mode, shift enable) is tied to during the functional mode of the chip.

The ports with their compliance value are added to a BSDL COMPLIANCE_PATTERNS statement.

Command Reference for Encounter RTL Compiler

Design for Test

`-custom_cell_directory string`

Specifies the path to the directory that contains the files describing the custom boundary cells. Each cell must be described as a Verilog module in its own file. The basename of the file must match the name of the cell in the module description.

`-design design`

Specifies the name of the design in which you want to insert boundary scan logic. This option is required if you have loaded multiple designs.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-dont_map`

Prevents the inserted boundary scan logic from being mapped to technology gates even if the design is already mapped.

`-exclude_ports ports`

Excludes the specified ports from being considered for boundary scan logic insertion.

`-functional_clocks ports`

Specifies the ports that are seen as clocks for ATPG. These ports include

- async set and reset ports
- clocks used in functional mode, but not in scan shift mode

`-jtag_macro_location instance`

Specifies a hierarchical instance into which to insert the JTAG macro. By default, the JTAG macro is inserted in the top-level design.

`-physical`

Specifies to build the boundary scan register using physical information to minimize its wire length.

The physical placement information is obtained from the DEF file read in with the `read_def` command

Note: This option is mutually exclusive to the `-pinmap` option.

`-pinmap_file file`

Specifies the name of the file containing the mapping between the design ports and the actual package pins. This mapping is also used to determine the boundary scan order.

Refer to [Pinmap File Format](#) for more information.

Command Reference for Encounter RTL Compiler

Design for Test

Note: This option is mutually exclusive to the `-physical` option.

`-power_on_reset {pin | port}`

Specifies the power-on-reset pin which will be connected to the `JTAG_POR` input pin on the `JTAG_Macro` subdesign. This connection is made when the boundary scan logic is inserted in the design.

Note: If the power-on-reset signal is applied to a top-level port, you also need to exclude this port from inclusion into the boundary scan register using the `-exclude_ports` option.

`-preserve_tdo_connection`

Preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

If you do not specify this option, the existing net connections will be broken and new net connections will be made during boundary scan insertion from the `JTAG_TDO` and `JTAG_ENABLE_TDO` pins to the from-core and tristate enable pins of the TDO pad cell, respectively.

Note: If you preserve the TDO connections, such that the net is driven by user logic other than a pre-instantiated JTAG macro, boundary scan insertion will insert a JTAG macro and leave its `JTAG_TDO` pin unconnected in the netlist.

`-preview`

Shows the potential changes, without making any modifications to the netlist.

`-tck port`

Specifies the port name of the driver for the test clock of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: TCK

`-tdi port`

Specifies the port name of the driver for the test data (scan) input of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: TDI

`-tdo port`

Specifies the port name of the test data (scan) output of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Note: An existing TDO port must have a tristate I/O pad.

Command Reference for Encounter RTL Compiler

Design for Test

Default: TDO

-tms port Specifies the port name of the test mode select input of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: TMS

-trst port Specifies the port name of the (asynchronous) test reset of the JTAG macro. Specify this option when the existing JTAG port does not use the standard name.

Default: TRST

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Inserting Boundary-Scan Logic](#)
- [Physical Boundary-Scan Insertion](#)
- [JTAG-Controlled Scan Modes](#)

Related constraints:

[define_dft_jtag_instruction_register](#) on page 649
[define_dft_jtag_macro](#) on page 651
[define_dft_shift_enable](#) on page 686
[define_dft_test_clock](#) on page 693
[define_dft_test_mode](#) on page 697

Affects these commands:

[compress_scan_chains](#) on page 601
[insert_dft_mbist](#) on page 743
[insert_dft_ptam](#) on page 751

Sets these attributes:

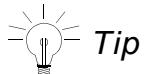
[boundary_type](#)
[dft_jtag_macro_exists](#)
[JTAG Port Attributes](#)

insert_dft compression_logic

```
insert_dft compression_logic [design]
    -use_existing_channels actual_scan_chains
    {-use_user_scan_chains scan_chains
    |-build_new_scan_chains integer}
    [-allow_shared_clocks] [-auto_create]
    [-decompressor {broadcast
        | xor [-spread_enable test_signal]}]
    [-master_control test_signal]
    [-compression_enable test_signal] [-target_period integer]
    [-compressor xor
        [-mask {wide1|wide2} [-mask_clock {port|pin}]}
        [-mask_load test_signal]
        [-mask_enable test_signal]
        [-mask_sharing_ratio integer]
        [-apply_timing_constraints
            [-timing_mode_names mode_list]]
        [-write_timing_constraints file]
        [-low_pin_compression
            [-lpc_control shift_enable]
            [-shift_enable shift_enable]]]
    | -compressor misr
        [-serial_misr_read]
        [-misr_observe test_signal]]
        [-misr_clock {port|pin}]
        {-misr_reset_enable test_signal
        | -misr_reset_clock test_signal}
        [-misr_read test_signal
        | -use_all_scan_ios_unidirectionally]
        [-misr_shift_enable test_signal]
        [-mask_sharing_ratio integer]
        [-mask {wide0 | wide1 | wide2}]
        [-mask_clock {port|pin}] [-mask_load test_signal]
        [-mask_enable test_signal_list]
    | -compressor hybrid
        [-serial_misr_read]
        [-misr_observe test_signal]
        [-misr_bypass test_signal]
        [-misr_clock {port|pin}]
        {-misr_reset_enable test_signal
        | -misr_reset_clock test_signal}
        [-misr_shift_enable test_signal]
        [-mask {wide0 | wide1 | wide2}]
        [-mask_sharing_ratio integer]
        [-mask_clock {port|pin}] [-mask_load test_signal]
        [-mask_enable test_signal_list] }
    [-dont_exploit_bidi_scanio] [-inside instance] [-preview]
    [-jtag_control_instruction jtag_instruction]
        [-allow_multiple_jtag_control]]
```

Inserts compression logic in a design that requires a fixed number of scan compression channels. It concatenates these channels into the specified number of full scan chains, and adds the information about the newly created fullscan chains to the `actual_scan_chains` directory while removing the original compression channels from the same directory.

Prerequisite: The design must have connected compression channels and these channels must appear in the `actual_scan_chains` directory. The channels should not be compressed (the `compressed` attribute must be `false`) and all elements in the compression channels must have passed the DFT rule checks. These compression channels should also not appear as scan chain definitions in the `scan_chains` directory.



Tip

The scan data input (sdi) pin and the scan data output (sdo) pin of the original compression channels will be disconnected but will be left in the netlist. It is the user's responsibility to remove them if desired.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-allow_multiple_jtag_control

When controlling the compression testmode from a JTAG macro, (that is, the `-jtag_control_instruction` option is specified), `compress_scan_chains` checks for the presence of other compression macros that are also JTAG controlled. RC-DFT does not automatically support such a configuration and therefore insertion of a second JTAG controlled compression macro is disallowed by default. Specify this option to bypass this check and insert additional JTAG controlled compression macros. When this option is specified, the tool assumes you will manually perform any additional stitching needed and will appropriately modify any files generated for Encounter Test.

Note: You must also specify the `-jtag_control_instruction` option with this option.

Command Reference for Encounter RTL Compiler

Design for Test

`-allow_shared_clock`

Allows the mask or MISR clock to be shared with an existing full-scan test clock.

Note: The mask or MISR clock can only be shared with a test clock if you added gating logic which prevents the scan flops from pulsing during the channel mask load or MISR reset sequences. Since functional clocks are typically used for scanning, this requirement means that the functional clocks must be gated during test.

Important

When specified with the `-auto_create` option, a `-mask_load` pin is automatically created. The `mask_load` pin must additionally be used to gate off the clock being shared. If this gating logic is not added, the mask loading or MISR reset procedure will corrupt the test data in the design.

`-apply_timing_constraints`

Applies timing constraints to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, no additional constraints are applied.

`-auto_create`

Automatically creates the necessary test pins as top-level ports.

If you omitted any of the following options

`-compression_enable`, `-spread_enable`, `-mask_clock`, `-mask_load`, `-mask_enable`, `-mask_or_misr_sdi`, `-mask_or_misr_sdo`, `-misr_clock`, `-misr_observe`, `-misr_reset_enable`, `-misr_read`, `-misr_bypass`, the appropriate ports will be created and named using the following format:

prefixOption

Command Reference for Encounter RTL Compiler

Design for Test

For example, `prefixcompression_enable`, where `prefix` is the value of the `dft_prefix` root attribute.

`-build_new_scan_chains integer`

Specifies the total number of top-level scan chains that is desired.

By default, the new ports will be called `SDI[i]` and `SDO[i]`, while the new chains will be called `CHAIN_i`, where `i` starts from 0 if no other ports or chains with those names exist.

`-compression_enable test_signal`

Specifies the name of the test signal that enables configuring the actual scan chains in compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. If you request to build the compression logic with a master control signal (`-master_control`), the input port driving the compression enable signal can be an existing functional pin (specified through the `-shared_in` option of `define_dft test_mode`). If you do not specify the `-master_control` option, you must define the compression enable signal without the `-shared_in` option.

`-compressor {xor | misr| hybrid}`

Specifies the type of compression logic to be built:

- `xor` specifies to build an XOR-based compressor
- `misr` specifies to build a MISR-based compressor
- `hybrid` specifies to build a MISR compression with MISR bypass capability. Bypassing the MISR allows you to perform compression using just the XOR compressor.

Default: xor

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- `xor` specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- `broadcast` specifies to build a broadcast-based decompression logic (simple scan fanout).

Command Reference for Encounter RTL Compiler

Design for Test

Default: broadcast

design Specifies the name of the top-level design whose scan chains must be compressed. You should specify this name in case you have multiple top designs loaded.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-inside *instance* Specifies the instance in which to instantiate the compression logic.

By default, the compression logic is inserted as a hierarchical instance in the top-level of the design.

-jtag_control_instruction *jtag_instruction*

Specifies which JTAG instruction will be used to target the compression macro's test data register.

-lpc_control *shift_enable_signal*

Specifies a shift-enable signal used to control low pin count compression.

You cannot specify the same shift-enable signal for both the **-lpc_control** and **-shift_enable** options.

If you omit this signal, the tool will use one of the default shift-enable signals.

If no shift-enable pin exists, the tool creates an **LPC_CONTROL** shift-enable pin if the **-auto_create** option is specified.

-low_pin_compression

Reduces the number of compression control pins required by using encoded control signals.

-mask {wide0 | wide1 | wide2}

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the **-compressor** option.

By default, no masking logic is inserted.

Note: The syntax indicates which types are available for each of the compressor types.

Command Reference for Encounter RTL Compiler

Design for Test

`-mask_clock {pin|port}`

Specifies the clock that controls the mask registers.

Note: The input port associated with this option can be an existing functional pin. This clock cannot be shared with an existing full-scan test clock pin unless you also specify the `-allow_shared_clocks` option.

`-mask_enable test_signal`

Specifies the name of the test signal that controls whether mask bits should be applied during the current scan cycle.

For `wide2` masking, two mask enable signals must be specified.

Note: If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-mask_load test_signal`

Specifies the name of the test signal that enables loading of the mask data into the mask data registers.

Note: If the `mask_clock` is dedicated (that is, is only used for mask register loading), this signal is not needed. If this signal is shared (is used to clock the MISR or other logic in the circuit), this signal is needed to gate non mask load clock pulses from corrupting the mask registers. If the `mask_clock` is shared with other logic, you can use this signal to protect the shared logic from corruption during the mask load sequence.

`-mask_sharing_ratio integer`

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the value specified for the compression ratio.

Note: This option is only valid with `wide1` and `wide2` masking.

`-master_control test_signal`

Specifies the master control signal that gates the compression enable signal used for compression.

Command Reference for Encounter RTL Compiler

Design for Test

Note: This test signal must be dedicated for test and must have been defined using the `define_dft test_mode` command.

`-misr_bypass test_signal`

Specifies the test signal used to bypass the MISR-based logic. This test signal is required in hybrid compression mode.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_clock {pin|port}`

Specifies the clock that controls the MISR registers.

Note: This input port cannot be shared with an existing full-scan test clock unless it is only used to accumulate the MISR signature or if the `-allow_shared_clocks` option is specified. The `-misr_clock` option is only used to accumulate the MISR signature if there are separate `-mask_clock` and `-misr_reset_clock` signals specified.

`-misr_observe test_signal`

Specifies the test signal used to select Serial MISR Read. This is required when the `-serial_misr_read` option is specified unless `-auto_create` or `-jtag_control_instruction` is also specified.

Note: You must also specify the `-serial_misr_read` option with this option.

`-misr_read test_signal`

Specifies the test signal to configure any bidirectional scan I/O pads for MISR compression.

Note: This option is mutually exclusive with the `-use_all_scan_ios_unidirectionally` option. Using scan I/O bidirectionally during MISR compression is only available with the `-compressor misr` option. When using the `-compressor hybrid` option, all scan I/O are used unidirectionally.

Command Reference for Encounter RTL Compiler

Design for Test

`-misr_reset_clock test_signal`

Specifies a separate dedicated test signal that is used to asynchronously reset the MISR.

Note: This option is mutually exclusive with the `-misr_reset_enable` option.

`-misr_reset_enable test_signal`

Specifies the test signal used to reset the MISR registers.

Notes:

- This option is mutually exclusive with the `-misr_reset_clock` option.
- If you do not specify the `-auto_create` option, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-misr_shift_enable test_signal`

Specifies the test signal used to enable MISR accumulation during scan shifting. When this signal is de-asserted, the contents of the MISR register will not change.

Note: If this option is omitted, the default shift-enable test signal for the design is used. If this option is specified, you must have defined this test signal using the `define_dft shift_enable` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft shift_enable` command).

`-preview`

Reports the requested ratio, the maximum original scan chain length, the maximum subchain length, and the number of internal scan channels that would be created without making modifications to the netlist. Use this option to verify your compression architecture prior to inserting the compression logic.

`-serial_misr_read`

Specifies to include support for reading MISR bits serially through the scan data pins.

`-shift_enable shift_enable`

Command Reference for Encounter RTL Compiler

Design for Test

Specifies the shift-enable signal to be used for low pin count compression.

If you omit this option, the tool will use the default shift-enable signal.

`-spread_enable test_signal`

Specifies the name of the test signal that enables applying the input test data to an XOR-based spreader network.

Use this option when `-decompressor` is set to `xor`.

Note: If you do not specify the `-auto_create` or `-jtag_control_instruction` options, this test signal must have been defined using the `define_dft test_mode` command. The input port driving this test signal can be an existing functional pin (specified through the `-shared_in` option of the `define_dft test_mode` command).

`-target_period integer`

Specifies a target clock period (in picoseconds) used to optimize the compression macro. If the value zero (0) is specified, synthesis is performed with a low effort compile, and without applying external (input/output delay) constraints.

Default: value for `test_clock` period of the scan chains being compressed

`-timing_mode_names mode_list`

Specifies the timing modes for which to generate the additional timing constraints that apply to the compression control signals.

The timing modes are taken into account when you specify the `-apply_timing_constraints` and `-write_timing_constraints` options.

Note: This applies only to multi-mode designs. Modes are created with the `create_mode` command.

`-use_all_scan_ios_unidirectionally`

Disables use of bidirectional scan I/O for a MISR-based compressor.

Note: This option is mutually exclusive with the `-misr_read` option.

`-use_existing_channels actual_scan_chains`

Specifies the names of the actual scan chains to be used as compression channels.

`-use_user_scan_chains scan_chains`

Specifies which scan chain definitions (from the *scan_chains* directory) to use when building the full scan chains.

In this case, the tool will use the port and chain names defined with the `define_dft scan_chain` commands.

`-write_timing_constraints file`

Specifies the file to which to write the timing constraints applied to the appropriate compression control signals to prevent the mapper from considering these paths for timing optimization. The timing constraints are not written if you specify the `-preview` option.

Timing constraints will be applied in all user-specified timing modes.

Note: If your design has multiple timing modes but you did not specify the `-timing_mode_names` option to list the timing modes for which to write the constraints, constraints for all modes are written to the file.

Examples

In the following examples, assume that the design has 55 existing scan chains, but only five top-level scan chains are desired.

- The following command inserts compression logic in the design and auto creates the compression channels.

```
insert_dft compression_logic -build_new_scan_chains 5 -auto_create
```

- In the following example, the user first defines the five top-level chains. Next these chains are used by the `insert_dft compression_logic` command.

```
rc:/> define_dft scan_chain -name DFTChain1 -sdi SI1 -sdo SO1 -create_ports
rc:/> define_dft scan_chain -name DFTChain2 -sdi SI2 -sdo SO2 -create_ports
rc:/> define_dft scan_chain -name DFTChain3 -sdi SI3 -sdo SO3 -create_ports
rc:/> define_dft scan_chain -name DFTChain4 -sdi SI4 -sdo SO4 -create_ports
rc:/> define_dft scan_chain -name DFTChain5 -sdi SI5 -sdo SO5 -create_ports
rc:/> llength [find / -actual_scan_chain *]
55
rc:/> insert_dft compression_logic -use_user_scan_chains \
[find / -scan_chain DFTChain*] -auto_create
```

Command Reference for Encounter RTL Compiler

Design for Test

```
rc:/> llength [find / -actual_scan_chain *]  
5
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Low Pin Count Compression Using Encoded Compression Signals](#)
- [Inserting Compression Logic in a Design that Requires a Fixed Number of Channels](#)

Affected by these commands: [connect_scan_chains](#) on page 620

Affects this command: [report_dft_chains](#) on page 788

Sets these attributes: [compressed](#)

[dft_compression_signal](#)

[dft_mask_clock](#)

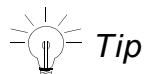
[dft_misr_clock](#)

[type](#)

insert_dft dfa_test_points

```
insert_dft dfa_test_points -input_tp_file string
  [-max_number_of_testpoints integer]
  [-min_slack integer]
  [-fault_threshold integer]
  [-test_control test_signal [-gate_clock]]
  [-test_clock_pin {pin/port} ]
  [-share_observation_flop integer]
  [-observe_only] [-verbose] [design]
```

Inserts selected test points identified by Encounter Test Deterministic Fault Analysis. Test points can be inserted that have minimal impact on the slack and which target a minimum specified fault count.



The `insert_dft dfa_test_points` command is recommended to be run with the design in default timing mode. Ensure that the design is in default timing mode before running this command by running the following commands:

```
set default_mode [filter default true [find / -mode *]] // to retrieve the
default timing mode
report timing -mode $default_mode
```

Options and Arguments

design

Specifies the name of the top-level design on which you want to perform test analysis and test-point selection.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-fault_threshold integer

Specifies to insert only those test point locations whose fault count is greater than or equal to the number specified.

Default: 0

-gate_clock

Enables clock gating for the test clocks of the inserted test points.

Note: To use this command option you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Command Reference for Encounter RTL Compiler

Design for Test

`-input_tp_file file`

Specifies the name of the file containing the test point locations.

The file is specified in Encounter Test format.

`-max_number_of_testpoints integer`

Specifies the number of test points to be inserted.

If this option is not specified, then all the test points from the file specified with the `-input_tp_file` option will be processed for insertion.

Default: All

`-min_slack integer`

Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: -10000000

`-observe_only`

Specifies to insert only observation test points. In this case, you do not need to specify a test control signal.

`-share_observation_flop integer`

Specifies the number of observation test nodes that can share an observation flop through an XOR tree.

Default: 1

`-test_clock_pin {port | pin}`

Specifies the test clock that drives the clock pin of the inserted test points during test mode operation. Specify a port or pin that drives the test clock.

If this option is not specified, the tool selects the test clock pin. For observe test points, the tool uses the test clock pin of the first flop in the fanout cone. For control test points, the tool uses the test clock pin of the first flop in the fanin cone. If the tool cannot find any test clock pin in the fanin or fanout cone, it uses the first test clock in the `dft/test_clock_domains` directory.

`-test_control test_signal`

Specifies the test signal to use to control the test points.

Note: You must have specified the test signal using the `define_dft test_mode` constraint.

`-verbose` Specifies to print test point details.

Examples

- The following command inserts all of the DFA test points, and when possible 3 test point locations will be shared through an XOR-tree to a common observation flop.

```
insert_dft dfa_test_points \
    -input_tp_file rc_et_dfa/myDesign.dfa.tpf \
    -test_control test_model \
    -test_clock_pin clk \
    -share_observation_flop 3
```

- The following example inserts the first 500 test points whose fault count is greater than or equal to 3 from the specified test point file `myfile`.

```
insert_dft dfa_test_points -max_number_of_testpoints 500 \
    -fault_threshold 3 -test_control tm -test_clock_pin clk -input_tp_file myfile
```

- The following example inserts all test points for test nodes with a minimum slack of 3ps, and will share 3 observation test nodes through an XOR tree from the specified test point file `myfile`.

```
insert_dft dfa_test_points -min_slack 3000 -share_observation_flop 3 \
    -test_control tm -test_clock_pin clk -input_tp_file myfile
```

- The following command inserts a maximum of 10 DFA test points whose nodes have a minimum slack of 1000 ps. Also, when possible, three test point locations will be shared through an XOR-tree to a common observation flop.

```
insert_dft dfa_test_points \
    -input_tp_file rc_et_dfa/myDesign.dfa.tpf \
    -min_slack 1000 \
    -test_control test_model \
    -test_clock_pin clk \
    -share_observation_flop 3 \
    -max_number_of_testpoints 10
```

- The following command inserts a maximum of 10 test points for locations whose fault count is greater than or equal to 15:

```
insert_dft dfa_test_points \
    -input_tp_file rc_et_dfa/myDesign.dfa.tpf \
    -test_control test_model \
    -test_clock_pin clk \
    -max_number_of_testpoints 10 \
    -fault_threshold 15
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Using Encounter Test to Perform a Deterministic Fault Analysis on a Scan Connected Netlist](#)
- [Inserting Scannable Test Points in Existing Scan Chains](#)

Affected by these constraints: [define_dft_test_clock](#) on page 693
 [define_dft_test_mode](#) on page 697

insert_dft jtag_macro

```
insert_dft jtag_macro [-design design]
    [-dont_map] [-jtag_macro_location instance]
    [-insert_without_pad_logic [-create_ports]
        [-dont_create_DFT_TDO_enable_port]]
    [-preserve_tdo_connection]
    [-tck port] [-tdi port] [-tdo port]
    [-tms port] [-trst port] [-power_on_reset pin/port]
    [-boundary_type {IEEE_11491|IEEE_11496}]
```

Inserts a JTAG Macro (if it is not already instantiated in the design).

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-boundary_type {IEEE_11491 | IEEE_11496}

Specifies the boundary scan architecture to use for the inserted JTAG Macro.

Default: IEEE_11491

-create_ports

Specifies whether to create the TAP ports if they do not exist.

If you do not specify the TAP signals using the -tdi, -tdo, -tms, -trst, and -tck options, the ports are named as *prefixtdi*, *prefixtdo*, *prefixtms*, *prefixtrst*, and *prefixtck*, where *prefix* is the value of the dft_prefix root attribute.

Note: Use this option with the -insert_without_pad_logic option.

-design *design*

Specifies the name of the design in which you want to insert the JTAG Macro. This option is required if you have loaded multiple designs.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-dont_create_DFT_TDO_enable_port

Prevents the creation of the block-level *prefixTDO_ENABLE* port which controls the tristate enable pin of the top-level TDO pad.

Command Reference for Encounter RTL Compiler

Design for Test

As a result, the JTAG_MODULE/JTAG_ENABLE_TDO output pin will be left unconnected in the netlist. You will need to connect this internal output pin to the appropriate logic to control the three-state enable pin of the JTAG TDO pad to pass boundary scan verification.

Note: Use this option with the
-insert_without_pad_logic option.

-dont_map
Prevents the inserted JTAG Macro from being mapped to technology gates even if the design is already mapped.

-insert_without_pad_logic
Specifies whether to insert the JTAG Macro into a design that does not have pad logic.

Unless you specify the
-dont_create_DFT_TDO_enable_port option, an additional port, *prefixTDO_ENABLE* will be created for the enable signal used to control the tristate pin of the top-level TDO pad.

-jtag_macro_location *instance*
Specifies a hierarchical instance in which to insert the JTAG Macro.

-power_on_reset {*pin* | *port*}
Specifies the power-on-reset pin name.

-preserve_tdo_connection
Preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

If you do not specify this option, the existing net connections will be broken and new net connections will be made from the JTAG Macro JTAG_TDO and JTAG_ENABLE_TDO pins to the from-core and tristate enable pins of the TDO pad cell, respectively.

Note: If you preserve the TDO connections, such that the net is driven by user logic other than the JTAG Macro, the JTAG Macro JTAG_TDO pin will be left unconnected in the netlist.

Command Reference for Encounter RTL Compiler

Design for Test

<code>-tck port</code>	Specifies the port name of the driver for the test clock of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.
	<i>Default:</i> TCK
<code>-tdi port</code>	Specifies the port name of the driver for the test data (scan) input of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.
	<i>Default:</i> TDI
<code>-tdo port</code>	Specifies the port name of the test data (scan) output of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.
	Note: An existing TDO port must have a tristate I/O pad.
	<i>Default:</i> TDO
<code>-tms port</code>	Specifies the port name of the test mode select input of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.
	<i>Default:</i> TMS
<code>-trst port</code>	Specifies the port name of the (asynchronous) test reset of the JTAG Macro. Specify this option when the existing JTAG port does not use the standard name.
	<i>Default:</i> TRST

Examples

- The following example inserts the JTAG Macro into the top-level netlist, creates TAP ports on the top-level netlist, and preserves the existing net connection to from-core and tristate enable pins of the TDO pad cell.

```
insert_dft jtag_macro -create_tap_ports design=design_name \
    -preserve_tdo_connection -insert_without_pad_logic
```

- The following example inserts the JTAG Macro into a specified hierarchical instance and connects it to TAP ports without pad logic.

```
insert_dft jtag_macro -inside instance_level design=design_name \
    -insert_without_pad_logic
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Inserting a JTAG Macro](#)
- [JTAG-Controlled Scan Modes](#)

Related constraints: [define_dft_jtag_instruction](#) on page 645

[define_dft_jtag_instruction_register](#) on page 649

Affects these commands: [compress_scan_chains](#) on page 601

[insert_dft_mbist](#) on page 743

[insert_dft_ptam](#) on page 751

Sets these attributes: [JTAG_PORT Attributes](#)

[dft_jtag_macro_exists](#)

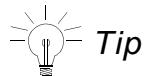
insert_dft lockup_element

```
insert_dft lockup_element  
  actual_scan_chain [actual_scan_chain]...  
  [-terminal_lockup]
```

Inserts a lockup element as needed in the specified analyzed scan chains.

Analyzed scan chains are chains whose connectivity is traced by the RC-DFT engine when you define them using the `-analyze` option of the `define_dft_scan_chain` command.

Use this command on mapped netlists whose existing (configured) scan chains were either created by hand or using a third-party DFT insertion tool.



Tip
The RC-DFT engine determines the type of lockup element to be inserted from the value of the `dft_lockup_element_type` design attribute.

Options and Arguments

`actual_scan_chain` Specifies the name of an analyzed scan chain.

`-terminal_lockup` Allows to add a terminal lockup element to the specified analyzed scan chains.

Example

The following example inserts lockup elements as needed in all analyzed scan chains.

```
insert_dft lockup_element [filter analyzed true \  
[find /designs/design/dft -actual_scan_chain *]]
```

Related Information

[Analyzing Chains in a Scan-Connected Netlist in Design for Test in Encounter RTL Compiler](#)

Affected by this attribute: [dft_lockup_element_type](#)

insert_dft logic_bist

```
insert_dft logic_bist
  -bist_clock_port [-bist_clock_hookup pin]
  [-add_buffer_to_bist_clock]
  [-capture_window_counter integer]
  [-program_counter integer]
  [-tester_available port...]
  [-inside instance] [design]
```

Inserts Logic Bist (LBIST) logic into the design.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-add_buffer_to_bist_clock

Adds a buffer to the BIST clock.

Specify this option when you use a top-level clock as LBIST clock to avoid having to manually specify the number of cycles required for LBIST when Encounter Test is used for simulation.

-bist_clock_port

Specifies the port to which the BIST clock is applied. You can specify either a top-level dedicated port or an oscillator reference clock.

-bist_clock_hookup pin

Specifies the hookup pin for the BIST clock.

Usually the PLL output is used as hookup for the BIST clock.

-capture_window_counter integer

Specifies the value to be loaded in the capture window counter at the start of the runbist operation.

Default: 7

design

Inserts the LBIST logic in the specified top-level design.

-inside instance

Specifies the instance in which to instantiate the LBIST logic.

By default, the LBIST logic is inserted as a hierarchical instance in the top-level of the design.

Command Reference for Encounter RTL Compiler

Design for Test

`-program_counter integer`

Specifies the value to be loaded in the program counter at the start of the runbist operation.

Default: 2

`-tester_available port`

Specifies the ports that are available on the tester and that therefore can be skipped for testpoint insertion. These ports are written out to the pinassign file if defined as test signal and test clocks.

Related Information

[Inserting LBIST Logic in Design for Test in Encounter RTL Compiler](#)

Affected by these constraints: [define dft test mode](#) on page 697

Related commands: [write et_lbist](#) on page 837

[write logic_bist macro](#) on page 850

insert_dft mbist

```
insert_dft mbist
  {-config_file config_file
   | -preview [-config_file config_file]}
  [-connect_to_jtag | -dont_create_mbist_ports
   |-direct_access_only [-dont_create_mbist_ports] ]
  [-dft_configuration_mode mode]
  [-test_control test_signal]
  [-interface_file_dirs string] [-dont_map]
  [-dont_check_dft_rules] [-dont_check_mbist_rules]
  [-diagnose_mbist string] [-diagnose_robust_mbist string]
  [-run_mbist string] [-raa_mbist string]
  [-repair_mbist string] [-read_mbist string]
  [-continue_mbist string]
  [-design string] [-module_prefix string]
  [-directory dir_path] [-measure_ports ports]
  [-mode mode_name [-notmode mode_name...]]]
```

Analyzes the design's memories to generate a configuration file template when the `-preview` option is used, optionally specifying a configuration file containing memory module information using the `-config_file` option, or inserts Memory Built-In-Self-Test (MBIST) or MBIST supporting logic based on the definitions in the configuration file specified using the `-config_file` or `-interface_file_dirs` options.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-config_file config_file`

Specifies the file that contains the user-defined configuration parameters which selects the MBIST features and controls for the insertion of the test logic for targeted memories.

You can use this option in conjunction with the `-interface_file_dirs` option as part of a bottom-up flow. The `-interface_file_dirs` option points to files that describe the MBIST structures that have been inserted earlier in a block of this design.

You can use this option with the `-preview` option to supplement the library information for the memory modules in the design. In this situation, only module statements are permitted in the configuration file.

Command Reference for Encounter RTL Compiler

Design for Test

<code>-connect_to_jtag</code>	Connects MBIST logic's JTAG interface pins to a pre-instantiated JTAG macro (either Cadence's or Third party). Use this option when the instance is the top level of the chip, and you have instantiated the JTAG macro and will use it to access the MBIST engines.
<code>-continue_mbist string</code>	Specifies the CONTINUE_MBIST instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines. <i>Default:</i> CONTINUE_MBIST
<code>-design design</code>	Specifies the name of the top-level design.
<code>-dft_configuration_mode mode</code>	Specifies the configuration mode in which MBIST will be operating. MBIST test pin verification and netlist back-tracing using the check_mbist_rules command must be done in this mode as well. <i>Default:</i> mbist
<code>-diagnose_mbist string</code>	Specifies the DIAGNOSE_MBIST instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines. <i>Default:</i> DIAGNOSE_MBIST
<code>-diagnose_rombist string</code>	Specifies the DIAGNOSE_ROMBIST instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines. <i>Default:</i> DIAGNOSE_ROMBIST
<code>-direct_access_only</code>	Inserts MBIST logic and makes the necessary connections using the MBIST direct access information only. In this case, any information for the JTAG macro is ignored even when a JTAG macro is specified.
<code>-directory directory_path</code>	Specifies the directory to which the interface files (MBIST pattern control and TDR mapping files) must be written. In a bottom-up flow, this directory is specified as input using the -interface_file_dirs option, when this command is run on a higher level of the hierarchy.

Command Reference for Encounter RTL Compiler

Design for Test

*Default: current_working_directory/
mbist_design*

-dont_check_dft_rules

Prevents the DFT rules from being automatically checked after MBIST insertion.

-dont_check_mbist_rules

Prevents the MBIST rules from being automatically checked after MBIST insertion.

-dont_create_mbist_ports

Prevents creation of MBIST control ports for subsequent JTAG macro and direct access connection at the top-level instance specified by the **-instance** option.

Use this option when the instance is the top level of the chip, and you have not yet instantiated the JTAG macro, or you use the direct access method and these ports were predefined.

-dont_map

Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

-interface_file_dirs string

Specifies a list of interface file directories which contain the MBIST pattern control and TDR mapping files for blocks in which MBIST logic has already been inserted. In a bottom-up flow, these files represent an abstract model of the BIST-ed blocks and are used by the **insert_dft mbist** command when processing the larger design. Separate the directory names with blank spaces.

-measure_ports ports

Specifies the tester-accessible ports to be used as bitmap pattern measure ports in addition to the default JTAG TDO port.

Note: This option applies to MBIST bitmap patterns and chip-level designs only.

-mode mode_name

Specifies the name of the timing mode for which you want to perform timing analysis on the MBIST logic only. The command adds timing constraints for the specified mode.

Command Reference for Encounter RTL Compiler

Design for Test

Do not use this option, if you want to perform timing analysis on MBIST in functional timing mode with the rest of the functional design.

Note: The specified mode should exist in the `/designs/design/modes` directory.

`-module_prefix string`

Specifies an additional character string to append to the default prefix `tem`. The string is placed on all modules created and inserted by the `insert_dft mbist` command.

Note: This option is required in case of an MBIST block-level insertion flow.

`-notmode mode_name` Specifies the name(s) of the timing mode(s) in which the MBIST logic should not be considered during timing analysis. You can only specify this option if you specified the `-mode` option.

Note: The specified mode should exist in the `/designs/design/modes` directory.

`-preview`

Requests the creation of a configuration file template without performing insertion. The template file can be used to review configuration file content or as a basis to further edit content.

This option can be used with the `-config_file` option to supplement the library information for the memory modules in the design. In this situation, only module statements are permitted in the configuration file.

`-raa_mbist string` Specifies the `RAA_MBIST` instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines.

Default: RAA_MBIST

`-read_mbist string` Specifies the `READ_MBIST` instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines.

Default: READ_MBIST

`-repair_mbist string`

Specifies the `REPAIR_MBIST` instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines.

Default: REPAIR_MBIST

Command Reference for Encounter RTL Compiler

Design for Test

-run_mbist string Specifies the RUN_MBIST instruction that must be loaded into the IEEE 1149.x TAP controller to access the MBIST engines.

Default: RUN_MBIST

-test_control test_signal

Indicates the user-defined test-control signal which must be held in its inactive state when the chip is in MBIST mode.

This signal will be asserted when the chip is in ATPG/SCAN mode to test MBIST logic for manufacturing defects.

Examples

- The following command analyzes the netlist to identify memory instances and generates an MBIST configuration file template for this design.

```
insert_dft mbist -preview
```

- The following command analyzes the netlist to identify memory instances and generates an MBIST configuration file template for this design with guidance on memory characteristics from the user-specified configuration file.

```
insert_dft mbist -preview -config_file \
..../et_inputs/my_module_configurations.txt
```

- The following command inserts the MBIST logic as described in the configuration file, by default, to the design module at the highest level of hierarchy of the design.

```
insert_dft mbist -config_file ..../et_inputs/my_configuration.txt
```

- The following command inserts the MBIST logic as described in the configuration file, into the design module `chip_top` of the design.

```
insert_dft mbist -config_file ..../et_inputs/my_configuration.txt \
-instance chip_top
```

- The following command inserts the MBIST logic as described in the configuration file, into design module `chip_top` placing interface files into a specified directory.

```
insert_dft mbist -config_file ..../et_inputs/my_configuration.txt \
-instance chip_top -directory ..../my_et_files
```

- The following command inserts the MBIST logic as described in the configuration file, into the design module `chip_top` of the design. The active high ScanTestMode signal will be deasserted during MBIST test operations.

```
define_dft test_mode -name ScanTestMode -active high \
[find ./des* -pin ScanTestMode]
```

```
insert_dft mbist -config_file ..../et_inputs/my_configuration.txt \
-instance chip_top -test_control ScanTestMode
```

Related Information

[Design Flows](#) in “Inserting Memory Built-In-Self-Test Logic” in *Design for Test in Encounter RTL Compiler*

Affected by these constraints:	<u>define dft dft configuration mode</u> on page 636 <u>define dft mbist clock</u> on page 656 <u>define dft mbist direct access</u> on page 659 <u>define dft test mode</u> on page 697
Affects these commands:	<u>insert dft boundary scan</u> on page 717 <u>insert dft jtag macro</u> on page 736 <u>write do lec</u> on page 238 <u>write sdc</u> on page 265
Related commands:	<u>check mbist rules</u> on page 595 <u>write dft rtl model</u> on page 821 <u>write et bsv</u> on page 830 <u>write et mbist</u> on page 840 <u>write mbist testbench</u> on page 852
Related attributes:	<u>dft rtl insertion</u> <u>mbist instruction set</u> <u>mbist interface files location</u>

insert_dft opcg

```
insert_dft opcg
  -opcg_enable test_signal
  -opcg_load_clock test_clock
  -scan_clock test_clock
  -test_enable test_signal
  [-preview] [design]
```

Inserts OPCG logic that generates on-chip launch and capture clocks for testing of at-speed delay defects.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-design design Specifies the name of the design for which to insert OPCG logic.

This option is only required if multiple designs are loaded.

If there is only one top-level design, you can omit the design name. In this case, the tool will use the top-level design of the design hierarchy.

-opcg_enable test_signal Specifies the test signal to be used to enable the OPCG mode.

The test signal must have been previously defined by a `define_dft test_mode` constraint.

-opcg_load_clock test_clock Specifies test clock that will be used to clock the side scan chains (scan chains in the OPCG logic).

The test signal must have been previously defined by a `define_dft test_clock` constraint.

-preview Shows the potential changes for the OPCG logic, without making any modifications to the netlist.

-scan_clock test_clock Specifies the test clock that will be used for shifting the OPCG logic in fullscan mode.

The test signal must have been previously defined by a
`define_dft test_clock` constraint.

`-test_enable test_signal`

Specifies the test signal to be used to gate the OPCG enable.

The test signal must have been previously defined by a
`define_dft test_mode` constraint.

Example

```
insert_dft opcg -opcg_enable OPCGMODE -opcg_load_clock opcg_load_clk \
-scan_clock scanclk -test_enable testmode
```

Related Information

[Inserting the OPCG Logic in Design for Test in Encounter RTL Compiler](#)

Related constraints:

[define_dft shift_enable on page 686](#)

[define_dft test_clock on page 693](#)

[define_dft test_mode on page 697](#)

insert_dft ptam

```
insert_dft ptam
  -power_test_enable {pin|port}
  [-power_test_enable_active {low|high} ]
  [-shift_enable test_signal ]
  [-instruction string] [-connect_to_jtag]
  [-directory string ] [-preview]
  [-dont_map ] [-dont_check_dft_rules ]
```

Inserts Power Test Access Mechanism (PTAM) logic to facilitate chip power management during test.

Note: Some files may require customization according to the setup requirements.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-connect_to_jtag

Connects the PTAM logic to the JTAG macro instance. The JTAG macro instance must first be created using either the `insert_dft boundary_scan` command or the `insert_dft jtag_macro` command. If `-connect_to_jtag` is not specified and a JTAG macro instance is detected in the current session, a warning message will be issued to notify of the JTAG macro instance's existence, otherwise there will be no attempt to connect to a JTAG macro instance.

-directory string

Specifies the directory to which the mode initialization and pin assign files are written.

Default: current_working_directory/ptam

-dont_check_dft_rules

Prevents the DFT rules from being automatically checked after PTAM insertion.

-dont_map

Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

Command Reference for Encounter RTL Compiler

Design for Test

`-instruction string`

Specifies the name of the instruction which must be loaded into the IEEE 1149.x TAP controller instruction register to access the PTAM test data register.

`-power_test_enable {pin | port}`

Identifies the power-test-enable pin or port. Asserting this pin to an active state will enable the PTAM logic to override the design's power manager control pins. This serves as a master mode control signal.

Note: This cannot be the same pin or port identified by `define_dft test_mode` that control pin sharing if the PTAM I/O are shared

`-power_test_enable_active {high | low}`

Specifies the active value for the power-test-enable pin or port.

Default: high

`-preview` Shows the potential changes without making any modifications to the netlist.

`-shift_enable test_signal`

Designates a shift-enable test signal used to override the PTAM gating logic of the power manager output control signals during scan test.

If you omit this option, the default shift-enable signal specified using `define_dft shift_enable` is used as selected by its `default_shift_enable` attribute.

Examples

- The following command inserts the PTAM logic into design `chip_top`.

```
insert dft ptam -instruction PTAM -power_test_enable /chip_top/PwrTe \
-directory ${workdir}/testmode_data
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Inserting Power Test Access Mechanism \(PTAM\) Logic in Design for Test in Encounter RTL Compiler](#)

Affected by these commands: [read_cpf](#) on page 922

[define_dft shift_enable](#) on page 686

[create_isolation_rule](#) in the *Common Power Format Language Reference*

[create_power_domain](#) in the *Common Power Format Language Reference*

[create_state_retention_rule](#) in the *Common Power Format Language Reference*

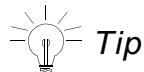
insert_dft_rrfa_test_points

```
insert_dft_rrfa_test_points
  { -input_tp_file file
    | [-atpg [-atpg_effort {low|medium|high}]
      [-atpg_options string]
      [-build_model_options string]
      [-build_testmode_options string]
      [-rrfa_effort {low|medium|high}]
      [-rrfa_options string]
      -directory string [-et_log file] [-verbose]
      [-output_tp_file file]
      [-max_number_of_testpoints integer]
      [-min_slack integer]
      [-share_observation_flop integer]
      [-library string]
      [-test_control test_signal [-gate_clock]
      [-observe_only]
      [-test_clock_pin {port|pin}] [design]
```

Invokes Encounter Test to

- Perform Automatic Test Pattern Generator (ATPG) based testability analysis to prune out the ATPG detectable faults (if the -atpg option is selected)
- Choosing the -atpg option does affect the runtime.
- Perform Random Resistance Fault Analysis (RRFA) based testability analysis and test-point selection
- Insert selected test points that have minimal impact on the slack

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).



The `insert_dft_rrfa_test_points` command is recommended to be run with the design in default timing mode. Ensure that the design is in default timing mode before running this command by running the following commands:

```
set default_mode [filter default true [find / -mode *]] // to retrieve the default timing mode

report timing -mode $default_mode
```

Options and Arguments

-atpg	Runs ATPG-based testability analysis to prune the ATPG detectable faults before running random-resistant fault analysis.
-atpg_effort {low medium high}	Specifies the effort to be used for the ATPG-based testability analysis. <i>Default:</i> low
-atpg_options <i>string</i>	Specifies extra options to run ATPG-based testability analysis in a string. For more information on these options, refer to the <code>create_tests</code> command in the <i>Command Line Reference</i> (of the Encounter Test documentation). Note: This option is mutually exclusive with the <code>-input_tp_file</code> option.
-build_model_options { <i>option1=value option2=value</i> }	Specifies extra options to apply when building the Encounter Test model. Note: For more information on these options, refer to the <code>build_model</code> command in the <i>Command Line Reference</i> (of the Encounter Test documentation).
-build_testmode_options { <i>option1=value option2=value</i> }	Specifies extra options to apply when building the test mode for Encounter Test. Note: For more information on these options, refer to the <code>build_testmode</code> command in the <i>Command Line Reference</i> (of the Encounter Test documentation).
<i>design</i>	Specifies the name of the top-level design on which you want to perform test analysis and test-point selection. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
-directory <i>string</i>	Specifies the working directory for Encounter Test.

Command Reference for Encounter RTL Compiler

Design for Test

Note: This option is only required when you run an RRFA-based analysis

-et_log file Specifies the name of the Encounter Test log file. This file will be generated in the specified directory.

Default: eta_from_rc.log

-gate_clock Enables clock gating for the test clocks of the inserted test points.

-input_tp_file file Specifies the name of the file containing the test point locations. The file is specified in Encounter Test format.

If you do not specify this option, the test point locations are read from

- The file specified with the **-output_tp_file** option if you also specified the **-rrfa** option
- The following file in the working directory if you did not specify any file:

TB/testresults/TestPointInsertion.ASSUMESCAN.expt.

Note: This option is mutually exclusive with **-atpg_options** and **-rrfa_options**.

-library string Specifies the list of Verilog structural library files.

You can specify the files explicitly or you can specify an *include* file that lists the files. You can also specify directories of Verilog files but you cannot reference directories in an include file.

For example, assume the following Verilog files are required:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

You can specify the files in either of the following ways:

1. Explicitly:

```
insert_dft rrfa_test_points -library "./padcells.v \  
./stdcells.v ./memories ./ip_blocks" ...
```

2. Using an include file.

```
insert_dft rrffa_test_points \
-library "include_libraries.v ./memories \
./ip_blocks" ...
```

where you created a file `include_libraries.v` with the following contents:

```
'include "./padcells.v"
'include "./stdcells.v"
```

Note: This option is only required when you invoke this command on a mapped netlist.

`-max_number_of_testpoints integer`

Specifies the number of test points to be inserted.

You must specify an integer value greater than 0 when attempting to insert test points from a file using the `-input_tp_file` option.

By default, all test points are inserted.

`-min_slack integer` Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: -10000000

`-observe_only` Specifies to insert only observation test points. In this case, you do not need to specify a test control signal.

`-output_tp_file file`

Specifies the output file generated by the RRFA-based analysis.

If you do not specify this option, the test point locations are written to the following file in the working directory:

```
TB/testresults/TestPointInsertion.ASSUMESCAN.expt .
```

`-rrfqa_effort {low | medium | high}`

Specifies the effort to be used for the RRFA-based analysis.

Default: low

`-rrfqa_options string`

Specifies the extra options to run RRFA-based testability analysis in a string.

For more information on these options, refer to the `analyze_random_resistance` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option is mutually exclusive with the `-input_tp_file` option.

`-share_observation_flop integer`

Specifies the number of observation test nodes that can share an observation flop through an XOR tree.

Default: 1

`-test_clock_pin {port | pin}`

Specifies the test clock that drives the clock pin of the inserted test points during test mode operation. Specify a port or pin that drives the test clock.

If this option is not specified, the tool selects the test clock pin. For observe test points, the tool uses the test clock pin of the first flop in the fanout cone. For control test points, the tool uses the test clock pin of the first flop in the fanin cone. If the tool cannot find any test clock pin in the fanin or fanout cone, it uses the first test clock in the `dft/test_clock_domains` directory.

`-test_control test_signal`

Specifies the test signal to use to control the test points.

Note: You must have specified the test signal using the `define_dft test_mode` constraint.

`-verbose`

Specifies to print test point details.

Examples

- The following example performs only RRFA-based testability analysis and creates a file `myfile` with the suggested test point locations.

```
insert_dft rrfa_test_points -output_tp_file myfile
```

- The following example inserts 10 test points from the specified test point file `myfile`.

```
insert_dft rrfa_test_points -max_number_of_testpoints 10 \
-test_control tm -test_clock_pin_clk -input_tp_file myfile
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example performs ATPG-based testability analysis followed by RRFA-based testability analysis. The command generates a report on the fault coverage in the log file, and stores the suggested test point locations in the `TB/testresults/TestPointInsertion.ASSUMESCAN.expt` file.
`insert_dft rrfa_test_points -atpg`

- The following example performs ATPG-based testability analysis, RRFA-based testability analysis, and inserts 10 test points. During RRFA-based testability analysis, test point locations are written to the `TB/testresults/TestPointInsertion.ASSUMESCAN.expt` file, while during test point insertion, they are read from this file.

```
insert_dft rrfa_test_points -atpg -max_number_of_testpoints 10 \
-test_control tm -test_clock_pin clk
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Using Encounter Test to Automatically Select and Insert Test Points](#)
- [Requirements](#) in “Inserting Logic Built-In-Self-Test Logic”
- [Inserting Scannable Test Points in Existing Scan Chains](#)

Affected by these constraints:

[define_dft test mode](#) on page 697
[define_dft test clock](#) on page 693

insert_dft scan_power_gating

```
insert_dft scan_power_gating
    -max_number_of_testpoints integer [-min_slack integer]
    {-test_control test_signal | -preview}
    [-report_virtual_scan_power [-power_mode mode]
        "clockFreq [flopTogglePercent]" -preview]
    [-input_tp_file file] [-output_tp_file file]
    [-dont_check_dft_rules] [design]
```

Inserts gating logic at selected flop outputs to minimize switching power during scan shift. This command must be run prior to building the actual scan chains in the design.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features](#) in *Design for Test in Encounter RTL Compiler*.

Options and Arguments

design Specifies the name of the top-level design on which to perform test point insertion.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-dont check dft rules

Prevents the DFT rules from being automatically checked after inserting scan power gating.

-input tp file *file*

Specifies the name of the file containing the test point locations.
The file is specified in Encounter Test format.

If you omit this option, the test point locations are read from:

- The file specified with the `-output_tp_file` option
 - The following file in the working directory if you did not specify any file:

TB/testresults/TestPointInsertion.ASSUMESCAN.expt.

Command Reference for Encounter RTL Compiler

Design for Test

`-max_number_of_testpoints integer`

Specifies the maximum number of test points to be inserted. Selection is based on the specified number of identified test points and the weight of the point resulting from logic cone analysis. The weight indicates the probability of a higher power level if toggling occurs.

`-min_slack integer` Limits the insertion of a test point to those nodes that have the specified minimum slack (in ps).

Default: 2000

`-output_tp_file file`

Specifies the name of the generated output file containing the recommended test points.

If you do not specify this option, the test point locations are written to the following file in the working directory:

`TB/testresults/TestPointInsertion.ASSUMESCAN.expt`

`-power_mode mode` Specifies the power mode for which the test power must be reported.

Specify this option with the `-report_virtual_scan_power` option when the design has multiple power modes.

`-preview` Reports the test points to be added, without modifying the design.

`-report_virtual_scan_power "clockFreq [flopTogglePercent]"`

Performs a *virtual* insertion of the test points into the design and measures their impact on the reduction of scan power. Use a floating value to specify the test clock frequency (in MHZ) and the flop-toggle percentage to use in the measurement. If the flop toggle percentage is not specified, it will default to 50% of the test clock frequency.

Note: This option is only valid with the `-preview` option.

`-test_control test_signal`

Specifies the test signal to enable the test point. This option is not required when the command is run with the `-preview` option.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode -scan_shift` constraint.

Related Information

[Gating Functional Paths to Reduce Scan Shift Power in Design for Test in Encounter RTL Compiler.](#)

Affected by these constraints: [define_dft shift_enable](#) on page 686

[define_dft test_mode](#) on page 697

insert_dft shadow_logic

```
insert_dft shadow_logic
  {-around instances [-test_control test_signal]
   {-mode bypass
    |-mode no_share -test_clock_pin {port|pin}
      [-rise |-fall]
    |-mode share -test_clock_pin {port|pin}
      [-fall | -rise] }
   [-exclude pins | -only pins] [-group pins]...
   [-balance]
   |-auto [-minimum_shadow_logic_pins integer]
     [-exclude_shadow_logic instances]
     [-test_control test_signal]
     [-test_clock_pin {port|pin}] }
  [-dont_map] [-preview] [design]
```

This command either

- Automatically inserts shadow logic around logic abstract and timing models (by adding observable flops in non-share mode) when you use the `-auto` option.
- Manually inserts bypass logic and scannable logic with or without register sharing when you use the `-around` option.

You can insert two basic types of DFT shadow logic around a particular instance: bypass and scannable logic. Each shadow logic flip-flop can implement one control point and one observation point at the same time.

If you want to share observation and control points, either by setting `-mode` to `share` or `bypass`, the following sharing criteria are observed:

- ❑ If you specify `-group`, the specified inputs and outputs are grouped together as indicated
- ❑ For the remaining inputs and outputs that are not listed with `-group`, the first input will share the flip-flop with or be connected to the first output, the second input with the second output, and so on. The order is that specified in the HDL interface declaration.

Note: Test points are added for unit-directional pins only. Bidirectional pins and pins associated with test clock objects are skipped.

Options and Arguments

`-around instances` Specifies the instances around which the DFT shadow logic must be inserted. Specify a hierarchical instance name.

Command Reference for Encounter RTL Compiler

Design for Test

-auto	Automatically inserts shadow logic around logic abstract and timing models (by adding observable flops in non-share mode).
-balance	Groups unmatched input and output pins to have a balanced number of groups.
<i>design</i>	Specifies the name of the top-level design in which to insert shadow logic. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
-dont_map	Prevents the inserted logic from being mapped even if the design is already mapped to the target library.
-exclude_pins	Prevents the specified pins from being considered for shadow logic insertion.
-exclude_shadow_logic { <i>instance...</i> }	Excludes automatic shadow logic insertion for the specified instances. You can only specify instances of blackboxes or timing models.
-group <i>pins</i>	Specifies the pins to group when also using -mode share or -mode bypass. Each group can have multiple input pins and multiple output pins. Format the groups as follows: { <i>input_i</i> ... <i>output_j</i> ...}
	Separate the pins by spaces. If you have more than one group, you must specify multiple -group options.
	Note: If -mode is set to bypass, each group must have at least one input and one output. Otherwise, the number of inputs or outputs can be equal or larger than zero.
-minimum_shadow_logic_pins <i>integer</i>	Limits shadow logic insertion to logic abstract or timing models that have more pins (inputs and outputs) than the specified value. <i>Default:</i> 10
-mode	Specifies the type of shadow logic to insert.
bypass	Implements bypass logic. If you specify this option, you must balance the number of inputs and outputs.

Command Reference for Encounter RTL Compiler

Design for Test

	<code>no_share</code>	Inserts one scannable observation test point per input and one scannable control test point per output.
	<code>share</code>	Pairs each input with an output and uses one scannable control and observation test point for each pair. If there are a different number of inputs and outputs, uses one scannable observe (or control) test point is used for each remaining input (or output).
<code>-only pins</code>		Restricts the pins to be considered for shadow logic insertion to the specified ones.
<code>-preview</code>		Shows the potential changes, without making any modifications to the netlist.
<code>[-rise -fall]</code>		Specifies the edge of the test clock that is active during test mode operation. These options are only valid in conjunction with <code>-test_clock_pin</code> . <i>Default:</i> <code>-rise</code>
<code>-test_clock_pin {port pin}</code>		Specifies the test clock that drives the clock pin of the shadow flip-flops. You can specify a port or pin that drives the test clock.
<code>-test_control test_signal</code>		Specifies the test signal to use to control DFT logic (the multiplexers after the controlling points). Note: You must have specified the test signal using either the <code>define_dft shift_enable</code> or <code>define_dft test_mode</code> constraint.

Examples

In the following examples, the logic before the ATPG-untestable module is not observable and the logic after it is not controllable. Following is the Verilog input code for the ATPG-untestable module and its instantiation:

```
module blackbox (i1,i2, o1,o2,o3)
  input i1,i2;
  output o1,o2,o3;
  ...
  blackbox U1 (.i1(n_1), .i2(n_2), .o1(n_3), .o2(n_4), .o3(n_5));
  ...

```

Command Reference for Encounter RTL Compiler

Design for Test

- Using the following examples, bypass logic is used to make the two inputs observable and the three outputs controllable. The first command pairs input i1 to output o1, and input i2 to output o3 (skipping o2). The second command pairs input i1 and output o2.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -mode bypass \
-exclude o2
insert_dft shadow_logic -around U1 -test_control my_TM -mode bypass \
-only {i1 o2}
```

- The following example uses scannable test points and shares these test points as control and observation points.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock_pin CK \
-mode share
```

- The following example uses scannable test points but does not share these test points for control and observation points.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock CK \
-mode no_share
```

- The following example uses scannable test points, shares these test points for control and observation points, and controls by grouping which pins share a common test point. More specifically, i1 and o2 share a test point, and i2 and o1. In addition, no control point is inserted for the net driven by U1/o3.

```
define_dft test_mode -name my_TM -active high TM
insert_dft shadow_logic -around U1 -test_control my_TM -test_clock CK \
-exclude o3 -mode share -group {i1 o2} -group {i2 o1}
```

- The following example automatically inserts shadow logic around all logic abstract and timing model instances.

```
rc:/> insert_dft shadow_logic -auto -test_control my_tm
INFO: Using test clock pin '/designs/data_ram_fj/ports_in/CK' for testpoint
insertion.
WARNING: pin 'U_BIST_CONTROLLER/I_BIST_WR_ENABLE_REG/CP' is skipped from
shadow DFT insertion since it is driven by a clock
WARNING: pin 'U_BIST_CONTROLLER/I_BIST_WR_ENABLE_REG/QN' is skipped from
shadow DFT insertion because it has no load
Total number of test points inserted: 6
....
WARNING: pin 'U_REAL_DATA_RAM3/CK' is skipped from shadow DFT insertion since
it is driven by a clock
```

```
Total number of test points inserted: 79
```

```
Mapping shadow DFT logic...
...
652
```

Note: The command returns the total number of test points inserted.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Inserting DFT Shadow Logic in Design for Test in Encounter RTL Compiler](#)

Affects these commands: [check_dft_rules](#) on page 589

[report_dft_registers](#) on page 790

Related constraints: [define_dft_shift_enable](#) on page 686

[define_dft_test_clock](#) on page 693

[define_dft_test_mode](#) on page 697

insert_dft test_point

```
insert_dft test_point -location {pin|port}...
    [-test_control test_signal [-gate_clock]]
    -type { control_0 | control_1 } |
        {{async_0 | async_1 | async_any
        | control_node -node {pin|port}
        | control_observe_0 | control_observe_1
        | control_observe_node -node {pin|port}
        | control_scan
        | observe_scan [-max_observe_share integer]
        | scan | sync_0 | sync_1 | sync_any }
        -test_clock_pin {pin|port} [-rise|-fall] }
    [-dont_map]
```

Allows you to manually specify a control or observation test point to be added to the design. Control test points always require the specification of a test-mode signal. Test points that use scannable flip-flops to observe or control a node always require a test-clock signal.

For all of the scannable test points, you need to run [check_dft_rules](#) after the test point is inserted.

The command returns the path name of the inserted test point when it is a flip-flop.



Important

You can only specify multiple locations when you request to insert a test point of type observe_scan. In this case, the tool builds a balanced XOR-tree with all the specified location pins. Additionally, you can control the maximum number of pin locations to be observed by the same observation flop by specifying the -max_observe_share option. If a test-control signal is also specified, the tool builds the XOR-tree after each input is AND-ed or OR-ed with the test-control signal. This prevents switching along the XOR-tree when not in test mode. If the test control is active high, gating happens by AND-ing, otherwise by OR-ing. The output of the last XOR-gate is fed to the D input of the observation flip-flop.

Options and Arguments

- | | |
|-------------|---|
| -dont_map | Prevents the inserted logic from being mapped even if the design is already mapped. |
| -gate_clock | Enables clock gating for the test clocks of the inserted test points. |

Command Reference for Encounter RTL Compiler

Design for Test

Note: To use this command option you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

`-location {port | pin}`

Specifies the location of the control point or observation point. Specify an existing hierarchical pin name or a top-level port. For observation test points, the pin can be an input or output pin. For control test points, the result is different depending on the direction of the location. See the [Examples](#) on page 771.

Note: You can only specify multiple locations for a test point of type `observe_scan`.

Note: If you specify a bidirectional pin, no logic will be inserted unless you specify the direction of the pin.

`-max_observe_share integer`

Specify the maximum number of locations to be shared for an observe test point.

Note: This option applies only when you set `-type` to `observe_scan`.

`-node {pin | port}` Specifies the pin or port to insert when `-type` is set to `control_node` or `control_observe_node` and when the signal specified by `-test_control` is active.

`[-rise | -fall]` Specifies the edge of the specified test clock that is active during test mode operation. These options are only valid in conjunction with `-test_clock`.

Note: You must use the same clock edge when inserting a control flip-flop and an observation flip-flop.

Default: `-rise`

`-test_clock_pin {port | pin}`

Specifies the test clock that drives the clock pin of the inserted flip-flops during test mode operation. You can specify a port or pin that drives the test clock.

This option is required for all type point types set using the `-type` option except those of type `control_0` or `control_1`.

Command Reference for Encounter RTL Compiler

Design for Test

`-test_control test_signal`

Specifies the test signal to use to control or observe the specified location point.

Note: You must have specified the test signal using either the `define_dft shift_enable` or `define_dft test_mode` constraint.

Note: Test points of type `observe_scan` do not require a test signal.

`-type`

Specifies the type of test point to insert at the specified location when the signal specified by `-test_control` is active.

Possible values are:

`async_0`

Inserts an asynchronous control test point that forces the control point to the value 0.

`async_1`

Inserts an asynchronous control test point that forces the control point to the value 1.

`async_any`

Inserts an asynchronous control test point that forces the control point to take either the original value or the inverted value.

`control_0`

Inserts a constant value 0.

`control_1`

Inserts a constant value 1.

`control_node`

Inserts an arbitrary node.

`control_observe_0`

Inserts a control and an observation point. The control point is forced to the value 0.

`control_observe_1`

Inserts a control and an observation point. The control point is forced to the value 1.

`control_observe_node`

Inserts a control and an observation point. The control point is forced to the value of the node specified by `-node`.

Command Reference for Encounter RTL Compiler

Design for Test

control_scan	Inserts a flip-flop to force a particular value at the specified location during test mode operation. The flip-flop must be remapped to a scan flop before connecting it to a scan chain later on. Note: This option requires you to specify the <code>-test_clock_pin</code> option.
observe_scan	Inserts a flip-flop to observe the specified location. The flip-flop must be remapped to a scan flip-flop before connecting it to a scan chain later on. This option requires you to specify the <code>-test_clock_pin</code> option.
scan	Inserts a scannable control and observation test point. Note: This option requires you to specify the <code>-test_clock_pin</code> option.
sync_0	Inserts a synchronous control test point that forces the control point to the value 0.
sync_1	Inserts a synchronous control test point that forces the control point to the value 1.
sync_any	Inserts a synchronous control test point that forces the control point to take either the original value or the inverted value.

Examples

- The following example inserts a scannable observation test point, using CLK to drive:

```
insert_dft test_point -location X/out -test_clock_pin CLK -type observe_scan
```
- The following example inserts a control-1 and scannable observation point:

```
insert_dft test_point -location X/out -test_control TM \
-test_clock_pin CLK -type control_observe_1
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following example inserts a scannable control point:

```
insert_dft test_point -location X/out -test_control TM \
-test_clock_pin CLK -type control_scan
```

- The following example inserts a scannable control and observation test point:

```
insert_dft test_point -location X/out -test_control TM \
-test_clock_pin CLK -type scan
```

- The following example inserts an async control-0 test point at hierarchical pin X/out:

```
insert_dft test_point -location X/out -test_control TM -type async_0 \
-test_clock_pin CK -fall.
```

- The following example inserts a synchronous control test point that forces the control point to the value 1 at hierarchical pin X/out:

```
insert_dft test_point -location X/out -test_control TM -type sync_1 \
-test_clock_pin CK -fall.
```

- The following example inserts two observation test points, one for pin1, pin2 and pin3, and the other for pin4, pin5 and pin6.

```
insert_dft test_point -type observe_scan -max_observe_share 3 \
-location pin1 pin2 pin3 pin4 pin5 pin6
```

Related Information

[Inserting a Control and Observation Test Point in Design for Test in Encounter RTL Compiler.](#)

Affects these commands: [check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

[synthesize](#) on page 348

Related constraints: [define_dft_shift_enable](#) on page 686

[define_dft_test_clock](#) on page 693

[define_dft_test_mode](#) on page 697

Related attributes: [Test Clock Attributes](#)

[Test Signal Attributes](#)

insert_dft user_test_point

```
insert_dft user_test_point -location {pin|port|subport}
  -cell {design|subdesign|libcell}
  {-cfi {pin|port}} | -no_cfi} [-cfo {pin|port}]
  -connect string [-connect string]...
  -name name
```

Inserts a user-defined test point at the specified location, and hooks it up to the specified pins.

Options and Arguments

-cell {*design|subdesign|libcell*}

Specifies the module or library cell to instantiate. The module can be loaded as a parallel design, or as a subdesign.

-cfi {*pin | port*} Specifies the cell functional input (CFI) pin or port name.

-cfo {*pin | port*} Specifies the cell functional output (CFO) pin or port name.

-connect *string* Specifies a string consisting of a cell pin and the corresponding source-signal pin to which the cell pin must be connected.

This string has the following format:

{*cell_pin source_pin*}

Use this option to specify most connections to the cell, except for the connections to the CFI and CFO pins.

-location {*pin|port|subport*}

Specifies a pin, port or subport that identifies where the test point must be inserted.

-name *name* Specifies the instance name to be given to the user-defined test point.

-no_cfi Specifies that the user test point cell has no CFI pin.

Example

- The following example inserts design MyUserTI in design top at the in1[2] input port. Port MyCFI will be connected to input port in1[2].

```
insert_dft user_test_point -location top/in1[2] -cell /designs/MyUserTI \
  -cfi MyCFI -cfo MyCFO -connect {MyShiftEn se} -connect {MyWRCK wck}
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Inserting a User-Defined Control and Observation Test Point](#)
- [Inserting Scannable Test Points in Existing Scan Chains](#)

Affects these commands:

[check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

[synthesize](#) on page 348

insert_dft wrapper_cell

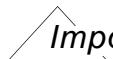
```
insert_dft wrapper_cell -location pin_list
  [-floating_location_ok]
  [-skipped_locations_variable Tcl_variable]
  [-shared_through {buffer|inverters_and_buffers|combinational}]
  [-wck pin] -wsen pin
  { -decoded_select_cfi pin
  | -wint {pin|port|constant}
    -wext {pin|port|constant} [-wcap pin] }
  [-guard {0|1} -wig pin -wog pin]
  [-dont_reuse_input_wrappers_for_output_ports]
  [-exclude pin_list] [-exclude_comb_feedthrough_paths]
  [-input_shared_threshold integer]
  [-output_shared_threshold integer]
  [-override_threshold_use_dedicated {pin|port}...]
  [-override_threshold_use_shared {pin|port}...]
  [-wrap_tied_constant_ports] [-inside_core]
  [-name segment_prefix] [-design design]
```

Selects a built-in IEEE 1500 standard wrapper cell based on the given specifications, inserts it at the specified location, and hooks it up to the specified control signals.

The wrapper cell logic is automatically identified as preserved wrapper-cell segments. You can use these segments in other segments (within nested segments) or specify the segments as elements when building the scan chains.

The command returns the directory path to the `scan_segment` objects that it creates. If multiple locations are specified, the command returns multiple segments. You can find the objects created by the `insert_dft wrapper_cell` in:

```
/designs/top_design/dft/scan_segments
```



Important

Any segment inserted by this command cannot be removed.

Options and Arguments

`-decoded_select_cfi pin`

Specifies the source pin name of the decoded control logic for the select-cfi signal.

Command Reference for Encounter RTL Compiler

Design for Test

	<p>Use this option to connect an already decoded signal. Otherwise, control decoder logic may have to be inserted for each cell, and extra control wires will have to be hooked up to each wrapper cell.</p>
<code>-design design</code>	Specifies the design in which you want to insert the wrapper cell. This option is required if multiple designs are loaded.
<code>-dont_reuse_input_wrappers_for_output_ports</code>	Allows to insert dedicated wrapper cells in the fanin of output ports that are being fed by wrapper cells previously inserted for input ports.
<code>-exclude pin_list</code>	Specifies the list of pins or ports to be excluded from wrapper insertion.
<code>-exclude_comb_feedthrough_paths</code>	Specifies whether to exclude pins or ports from wrapper insertion if combinational logic is found on the path from input to output.
<code>-floating_location_ok</code>	Specifies to insert a wrapper cell even if the specified pin (location) is floating.
<code>-guard {0 1}</code>	Specifies the guard for safe_value out of a wrapper cell. The safe value prevents testing of one block from interfering with another block. If this option is not specified, no guard logic will be included in the wrapper cell.
<code>-input_shared_threshold integer</code>	Specifies the maximum number of scannable flops that can be shared in wrapper cells. When the number of scannable flops exceeds the specified threshold value, a single dedicated wrapper cell will be inserted for the input port. <i>Default:</i> 10
<code>-inside_core</code>	Inserts the wrapper cells associated with hierarchical pins inside the core module.

Command Reference for Encounter RTL Compiler

Design for Test

`-location pin_list`

Specifies one or more pins that identify where the wrapper cell must be inserted.

Note: When specifying the pins of a blackbox instance use the RC pin name to identify the input or output pins.

`-name segment_prefix`

Specifies the segment name prefix.

If you specified a single location pin, and there is no name conflict, the segment name will correspond to the specified prefix, otherwise a unique name will be generated for each segment that is derived from the specified prefix.

`-output_shared_threshold integer`

Specifies the maximum number of scannable flops that can be shared in wrapper cells. When the number of scannable flops exceeds the specified threshold value, a single dedicated wrapper cell will be inserted for the output port.

Default: 10

`-override_threshold_use_dedicated {pin|port}...`

Specifies the list of pins or ports for which a dedicated wrapper cell must be inserted.

`-override_threshold_use_shared {pin|port}...`

Specifies the list of pins or ports for which a shared wrapper cell must be inserted.

`-shared_through {buffer|inverters_and_buffers|combinational}`

Specifies whether the functional flop in the wrapper cell must be shared. If a shared cell is inserted, the command traces through the logic to identify a shareable functional flop (or flops).

A functional flop in a wrapper cell can be shared if

1. It is directly connected to the core pin through
 - a series of buffers (buffer)
 - a series of inverters and buffers
 - complex combination logic (combinational)

Command Reference for Encounter RTL Compiler

Design for Test

2. It is mapped to a scan flip-flop for DFT purposes.
3. The clock pin to the flop is controllable; that is, the flip-flop must pass the DFT rules.
4. The flop has no connected enable pin.
5. The functional flop is not already shared with another wrapper cell.

Note: If you use this option and the functional flop cannot be shared, the RC-DFT engine issues a message and inserts a dedicated cell if you specified the `-wck` option, otherwise an error message is given.

`-skipped_locations_variable Tcl_variable`

Writes the locations where no wrapper cells can be inserted to the specified Tcl variable. If you omit this option, the command fails if you have any such locations specified.

`-wcap driver`

Specifies the capture control source pin or port name.

Note: This option is ignored if you specified the `-decoded_select_cfi` option.

`-wck driver`

Specifies the test clock source pin or port name. If specified, this pin will connect to the clock pin of the inserted dedicated wrapper cells.

Note: If this option is not specified, the tool will identify a local test clock to be used to connect to the clock pin of the inserted dedicated wrapper cells.

`-wig driver`

Specifies the in-guard control source pin or port name for an inward-facing wrapper cell.

Note: This signal will typically be controlled by the `-wext` control signal.

`-wint {pin|port|constant}`

Specifies the control source pin or port name, or constant value for inward-facing test mode.

Note: This option is ignored if you specified the `-decoded_select_cfi` option.

Command Reference for Encounter RTL Compiler

Design for Test

`-wext {pin|port|constant}`

Specifies the control source pin or port name, or constant value for outward facing test mode.

Note: This option is ignored if you specified the `-decoded_select_cfi` option.

`-wog driver`

Specifies the out-guard control source pin or port name for an outward-facing wrapper cell..

Note: This signal will typically be controlled by the `-wint` control signal.

`-wrap_tied_constant_ports`

Specifies to insert dedicated wrapper cells on output ports driven by a logic 0 or logic 1 constant.

`-wsen driver`

Specifies the shift-enable source pin or port name.

Examples

- The following example inserts a dedicated wrapper cell for port `in[0]`.

```
insert_dft wrapper_cell -location in[0] -wsen WSEN \
-wint WINT -wck WcIk1
```

- The following example inserts four dedicated wrapper cells external to the hierarchical pins of the blackbox instance `hardmacro`. The two wrapper cells inserted for the hierarchical input pins are controlled in `INTEST` mode, while the two wrapper cells inserted for the hierarchical output pins are controlled in `EXTEST` mode. Because the `hardmacro` instance is modeled as a blackbox, you must specify the `-floating_location_ok` option to insert the wrapper cells.

```
insert_dft wrapper_cell -wSEN WSEN -wCK WCK -wINT WINT -wEXT WEXT \
-floating_location_ok -location "hardmacro/pins_in/A hardmacro/pins_in/B \
hardmacro/pins_out/X hardmacro/pins_out/Y"
```

Related Information

[Inserting the Wrapper Cells in Design for Test in Encounter RTL Compiler](#)

Affects this command: [connect_scan_chains](#) on page 620

Sets this attribute: [core_wrapper](#)

insert_dft wrapper_mode_decode_block

```
insert_dft wrapper_mode_decode_block [-name string]
[-inside {instance|design}]
[-create_wrapper_shift_enables_for_delay_test
  shift_enable]
[-directory directory] [-preview] [-design design]
```

Builds a 1500 mode decode block based on the scan modes that were defined with type `wrapper` for use with the IEEE 1500 core wrapper cells insertion. This module decodes the various modes needed for different scan configurations and also decodes the EXTEST and INTEST signals from these scan modes.

Options and Arguments

-create_wrapper_shift_enables_for_delay_test *shift_enable*

Specifies the shift-enable signal to use to create two new shift enable signals: the wrapper shift-enable signal used for INTEST mode (gated by the decoded INTEST signal), and the wrapper shift-enable signal used for EXTEST mode (gated by the decoded EXTEST signal).

-design *design* Specifies the design in which you want to insert the wrapper mode decode block. This option is required if multiple designs are loaded.

-directory *directory*

Specifies the directory to which the generated decoder RTL file must be written.

Default: `current_working_directory/1500`

-inside *instance*

Specifies the hierarchical instance in which the wrapper mode decode block must be instantiated.

By default, the wrapper mode decode block is inserted in the top-level design.

-name *name*

Specifies the name of wrapper mode decode block.

Default: `wrapperModeDecodeBlock`

-preview

Prints the RTL of the generated decode block to the screen.

Example

The following example inserts a wrapper mode decode block based on the defined wrapper dft_configuration_modes.

```
define_dft dft_configuration_mode -name functional -type wrapper -usage mission \
    -mode_enable_low TS1 TS2 TS3 TS4
define_dft dft_configuration_mode -name serialIntest -type wrapper -usage intest \
    -mode_enable_low TS4 TS3 TS2 -mode_enable_high TS1
define_dft dft_configuration_mode -name serialExtest -type wrapper -usage extest \
    -mode_enable_low TS4 TS3 TS1 -mode_enable_high TS2
define_dft dft_configuration_mode -name parallelIntest -type wrapper \
    -usage intest -mode_enable_low TS4 TS3 -mode_enable_high TS2 TS1
define_dft dft_configuration_mode -name parallelExtest -type wrapper \
    -usage extest -mode_enable_low TS4 TS2 TS1 -mode_enable_high TS3
define_dft dft_configuration_mode -name parallelCompress -type wrapper \
    -usage intest -mode_enable_low TS4 TS2 -mode_enable_high TS3 TS1
insert_dft wrapper_mode_decode_block \
    -create_wrapper_shift_enables_for_delay_test DFTWSE
```

Related Information

[Inserting the Wrapper Mode Decoder in Design for Test in Encounter RTL Compiler](#)

Affects these commands:

[concat_scan_chains](#) on page 615

[connect_scan_chains](#) on page 620

Affected by this command:

[define_dft dft_configuration_mode](#) on page 636

map_mbist_cgc_to_cgic

```
map_mbist_cgc_to_cgic
  [-design design]
  [-clock_gating_cell libcell]
```

Maps the clock-gating logic inserted by the MBIST application to the specified integrated clock-gating cell.

Use this command if mapping was prevented during MBIST insertion.

Options and Arguments

`-clock_gating_cell libcell`

Specifies the name of a clock-gating cell whose `clock_gating_integrated_cell libcell` attribute value should equal `latch_posedge_precontrol`.

Note: If you omit the `-clock_gating_cell` option, the tool checks if the `lp_clock_gating_cell` attribute is specified on the module containing the clock-gating logic. If this libcell has the correct type, it will be used to replace the clock-gating logic, otherwise the tool will try to find the proper type of libcell inside the appropriate library domain to do the mapping. If you specify the wrong clock-gating cell type, a warning message will be issued and no clock-gating logic will be replaced.

`-design design`

Specifies the name of the top-level design.

Related Information

[Mapping Clock-Gating Logic Inserted by insert_dft mbist to Clock-Gating Integrated Cells in Design for Test in Encounter RTL Compiler](#)

Related command:

[insert_dft mbist](#) on page 743

read_dft_abstract_model

```
read_dft_abstract_model
  [-ctl [-use_scan_structures_se_only]] [-segment_prefix string]
  [-instance instance]
  [-assume_connected_shift_enable] file
```

Reads in the scan abstract model of a design that is used as a core or IP block in the current design. The scan abstract model defines the scan chain architecture of the subdesign and is used as scan chain segments in the configuration of the top-level scan chains of the current design.

The extracted scan chain information is stored in:

```
/designs/top_design/dft/scan_segments
```

Options and Arguments

-assume_connected_shift_enable

Indicates that the shift-enable port specified in the DFT abstract model for the block being read in is already connected to logic external to this block. Therefore the scan configuration engine does not need to modify the existing connection.

Note: If you specify this option and the shift-enable pin is *not* connected, the scan configuration engine will *not* make the connection.

If you do not specify this option, the scan configuration engine will make the connection to the shift-enable port specified in the DFT abstract model. If a connection already existed, it will be first removed.

-ctl

Specifies that the scan abstract model was written using the Core Test Language (CTL) format (IEEE format P1450 . 6).

If you omit this option, the scan abstract model is assumed to consist of a list of define_dft_abstract_segment commands, one scan segment per scan chain in the subdesign.

file

Specifies the file that contains the abstract model description.

-instance *instance*

Applies the scan abstract model to the specified hierarchical instance.

Command Reference for Encounter RTL Compiler

Design for Test

If you read in an abstract model written in native RC format, this instance must be an instantiation of the subdesign specified through the `-module` option in the abstract model.

If you read in an abstract model written in CTL format, this instance must be an instantiation of the subdesign specified in the `Environment` section of the CTL file.

If this option is omitted, the scan abstract model is applied to all instances of the subdesign.

`-segment_prefix string`

Adds the specified string as a prefix to the

- Segment name defined in the native RC format file
- Chain name defined in the CTL file

`-use_scan_structures_se_only`

Indicates that the shift-enable signal for an abstract segment must be read from its `ScanChain` definition in the `ScanStructures` block in the CTL file.

If a shift-enable signal is not specified, the abstract segment is created using the `-connected_shift_enable` option.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Set Up for DFT Rule Checker](#)
- [Hierarchical Compression Flow](#)
- [Bottom-Up Test Synthesis Flow](#)
- [Analyzing Chains in a Scan-Connected Netlist](#)

Affected by these commands: [check_dft_rules](#) on page 589

[connect_scan_chains](#) on page 620

[synthesize](#) on page 348

Related commands: [write_dft_abstract_model](#) on page 818

[write_hdl](#) on page 255 (-abstract)

Sets these attributes: [Scan Segment Attributes](#)

read_io_speclist

```
read_io_speclist iospeclist_file
```

Reads in the specified IOSpecList input file to be used for boundary scan insertion.

The IOSpecList input file is only required to provide information that cannot be inferred from the design, and the command-line options of the `insert_dft boundary_scan` command.

You need an IOSpecList input file if

- The I/O pad cells in your library do not use the standard pin names
- Your design has pin sharing logic to shared functional output signals that was inserted before you insert boundary scan logic
- You want to customize the location of the boundary cells in the boundary register

You can also use an IOSpecList input file if

- You want to use custom boundary cells
- You want to use user-defined TAP instructions (such as those required for MBIST or PTAM) and use specific opcodes specified using JTAG_Inline syntax

Note: You can also use the `define_dft jtag_instruction` command to enter user-defined instructions.

Options and Arguments

iospeclist_file Specifies the IOSpecList input file.

Related Information

[Reading an IOSpecList File in Design for Test in Encounter RTL Compiler](#)

Affects these commands:

[insert_dft boundary_scan](#) on page 717

[write_io_speclist](#) on page 848

replace_opcg_scan

```
replace_opcg_scan -edge_mode test_signal  
      [-dont_map] [design]
```

Replaces domain blocking scan flops with their OPCG-equivalent flops.

Options and Arguments

design Specifies the design in which you want to replace domain-blocking scan flip-flops.

-dont_map Prevents the inserted logic from being mapped even if the design is already mapped to the target library.

-edge_mode *test_signal*
Specifies the global edge-mode signal to connect.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- Inserting the Toggle Muxes
 - Top-Down Test Synthesis Flow with OPCG Logic Insertion
 - Requirements in “Inserting Memory Built-In-Self-Test Logic”

Affected by these commands: [reset_opcg_equivalent](#) on page 795
[set_opcg_equivalent](#) on page 799

replace_scan

```
replace_scan [-to_non_scan]  
           [-dont_check_dft_rules] [design]
```

This command either

- Replaces non-scan flops with their scan-equivalent flip-flops if the design was previously mapped.
In this case, the `dft_scan_map_mode` design attribute must be set to either `tdrc_pass` or `force_all`. If set to `tdrc_pass`, you must have run the DFT rule checker.
 - Replaces all scan flops that are part of shift register segments with non scan flops except for the first element of each shift register segment.

Options and Arguments

design Specifies the design in which you want to replace regular flip-flops.

-dont_check_dft_rules
Prevents the DFT rules from being automatically checked.

-to_non_scan Replaces all scan flops that are part of shift register segments to non scan flops except for the first element in the segments.

Related Information

See the following sections in *Design for Test in Encounter RTI Compiler*.

- Defining Scan-Equivalency between Non-Scan and Scan Flops to Map to Scan
 - Controlling Mapping to Scan in a Mapped Netlist
 - Identifying Shift Registers in a Mapped Netlist before Creating the Scan Chains

Affected by these commands: check dft rules on page 589

identify test mode registers on page 712

identify test mode registers on page 712

set scan equivalent on page 801

Affected by this attribute: dft scan map mode

report dft_chains

Refer to [report dft_chains](#) in the [Chapter 8, “Analysis and Report.”](#)

report dft_core_wrapper

Refer to [report dft_core_wrapper](#) in the [Chapter 8, “Analysis and Report.”](#)

report dft_registers

Refer to [report dft_registers](#) in the [Chapter 8, “Analysis and Report.”](#)

report dft_setup

Refer to [report dft_setup](#) in [Chapter 8, “Analysis and Report.”](#)

report dft_violations

Refer to [report dft_violations](#) in Chapter 8, “Analysis and Report.”

report scan_compressibility

Refer to [report scan_compressibility](#) in [Chapter 8, “Analysis and Report.”](#)

report test_power

Refer to [report test_power](#) in the [Chapter 8, “Analysis and Report.”](#)

reset_opcg_equivalent

```
reset_opcg_equivalent [libcell]...
```

Removes the specified scan library cells from the OPCG-equivalency table which was previously defined using a (number of) `set_opcg_equivalent` command(s).

If you do not specify any library cells, the command removes all OPCG-equivalent mappings.

Options and Arguments

<i>libcell</i>	Specifies a scan library cell to be removed from the OPCG-equivalency table.
----------------	--

Example

The following example removes the `snl_ffqx1` cell from the OPCG-equivalency table.

```
reset_opcg_equivalent snl_ffqx1
```

Related Information

Affects this command: [replace_opcg_scan](#) on page 786

Related command: [set_opcg_equivalent](#) on page 799

reset_scan_equivalent

```
reset_scan_equivalent [libcell]...
```

Removes the specified non-scan library cells from the scan-equivalency table which was previously defined using a (number of) [set_scan_equivalent](#) command(s).

If you do not specify any library cells, the command removes all scan-equivalent mappings.

Options and Arguments

<i>libcell</i>	Specifies a non-scan library cell to be removed from the scan-equivalency table.
----------------	--

Example

The following example removes the `snl_ffqx1` cell from the scan-equivalency table.

```
reset_scan_equivalent snl_ffqx1
```

Related Information

Affects this command: [replace_scan](#) on page 787

Related command: [set_scan_equivalent](#) on page 801

set_compatible_test_clocks

```
set_compatible_test_clocks
  {-all | -none | list_of_test_clocks}
  [-dont_check_dft_rules] [-design design]
```

Specifies the compatible test clocks whose related scan flip-flops can be merged into a single scan chain using lockup elements in between. By default, no test clocks are assumed compatible unless they are defined as independent test clocks in the same test clock domain.

Note: This command applies only to the muxed scan style.

Test clocks that are declared compatible belong to the same DFT clock domain.



Only those test clocks with the same clock period can be made compatible as independent test clocks in the same test clock domain.

Options and Arguments

<code>-all</code>	Specifies that all test clocks are compatible.
<code>-design <i>design</i></code>	Specifies the design for which you want to specify compatible test clocks.
<code>-dont_check_dft_rules</code>	Prevents the DFT rules from being checked automatically after specifying the compatible test clocks.
<code><i>list_of_test_clocks</i></code>	Specifies the compatible test clocks. You must specify the test clock object name. Note: To allow combining flip-flops from the same DFT domain—which are triggered by either edge of the same test clock—on the same scan chain, you need to set the <code>dft_mix_clock_edges_in_scan_chains</code> root attribute to <code>true</code> . By default, only same edge clocks are mixed.
<code>-none</code>	Specifies that none of the test clocks are compatible.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Mixing Different Test Clocks in the Same Scan Chain](#)
- [Concatenating Scan Chains](#)

Affects this command: [connect_scan_chains](#) on page 620

Related attribute: [dft_mix_clock_edges_in_scan_chains](#)

set_opcg_equivalent

```
set_opcg_equivalent
  -scan_cell libcell -opcg_cell libcell
  [-tieoff_pins string] [-pin_map list_of_pin_groups]
  -edge_mode_pin string -loop_back_pin string
```

Controls the OPCG-equivalent cell type that is used during the conversion of a scan flip-flop to an OPCG cell by the replace_opcg_scan command.

The command creates an OPCG-equivalency table.

An OPCG cell is a special scannable cell with an additional (hold) mux in its scan-data path. Depending upon the value of the -edge_mode_pin, the mux will either circulate or loopback the inverted value of its output pin to the -loop_back_pin, or capture the scan-in data from the scan-data path.

Options and Arguments

-edge_mode_pin *pin* Specifies the edge mode pin of the OPCG cell to connect to.

-loop_back_pin *pin* Specifies the loopback pin of the OPCG cell to connect to.

-opcg_cell *libcell* Specifies the OPCG library cell to map to.

-pin_map *list_of_pin_groups*

Indicates how to map a pin from the scan flop to a pin in the OPCG cell when the pin names in the cells do not match.

The *list_of_pin_groups* has the following format:

```
{ {scan_pin opcg_pin} {scan_pin opcg_pin} ... }
```

-scan_cell *libcell* Specifies a scan flip-flop library cell to be replaced.

-tieoff_pins *string* Specifies the tie-off value for extra pins on the OPCG cell.

The *string* has the following format:

```
{ {pin tie_off_value} {pin tie_off_value} ... }
```

The value can be a logic 0 or 1.

Example

The following example assumes that the pin names in the scan and OPCG flip-flops match, and that there are no extra pins in the OPCG flop to be tied off.

```
set_opcg_equivalent -scan_cell SDFFQ_X1M -opcg_cell S2DFFQQN_X1M \
    -edge_mode_pin TEL -loop_back_pin TI
```

Related Information

[Inserting the Toggle Muxes in Design for Test in Encounter RTL Compiler](#)

Affects this command: [replace_opcg_scan](#) on page 786

Related command: [reset_opcg_equivalent](#) on page 795

set_scan_equivalent

```
set_scan_equivalent  
  -non_scan_cell libcell -scan_cell libcell  
  [-tieoff_pins string] [-pin_map list_of_pin_groups]
```

Controls the scan-equivalent cell type that is used during the conversion of a non-scan flip-flop which passes the DFT rule checks to a scan flop. Use the `replace_scan` command to perform the actual conversion to scan.

Note: The RC-DFT engine automatically derives the scan data input, scan data output, and other test signals from the `test_cell` description of the scan flop in the target library.

Options and Arguments

`-non_scan_cell libcell`

Specifies a non-scan flip-flop library cell.

`-pin_map list_of_pin_groups`

Indicates how to map a pin from the non-scan flop to a pin in the scan flop when the pin names in the cells do not match.

The `list_of_pin_groups` has the following format:

```
 {{non_scan_pin scan_pin} {non_scan_pin  
 scan_pin} ...}
```

`-scan_cell libcell`

Specifies a scan flip-flop library cell.

`-tieoff_pins string`

Specifies the tie-off value for extra scan cell pins.

The `string` has the following format:

```
 {{pin value} {pin value} ...}
```

The value can be a logic 0 or 1.

Example

The following example assumes that the pin names in the non-scan and scan flip-flops match, and that there are no extra pins in the scan flop to be tied off.

```
set_scan_equivalent -non_scan_cell snl_ffqx1 -scan_cell snl_sffqx1
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Defining Scan-Equivalency between Non-Scan and Scan Flops to Map to Scan](#)
- [Controlling Mapping to Scan in a Mapped Netlist](#)

Affects this command: [replace_scan](#) on page 787

Related command: [reset_scan_equivalent](#) on page 796

update_scan_chains

```
update_scan_chains  
  [-flops instances] [-chains actual_scan_chains]  
  [-type {dfa|rrfa|user}] [-max_print_flops integer]  
  [-preview] [design] [> file]
```

Includes the identified scannable test points in the existing scan chains. These test points were inserted in the design after the scan chains were connected.

Note: Test points can only be added to scan chains that have not been compressed.

Options and Arguments

-chains *actual_scan_chains*

Specifies the names of the actual scan chains to be updated.
By default, all actual scan chains are updated.

design

Specifies the design in which you want to update the scan chains.

-flops *instances*

Specifies the test point instances to be added.

-max_print_flops *integer*

Specifies the maximum number of test point flops to report.
By default, all test points will be reported.

-preview

reports what would be done, but does not update the scan chains

-type {dfa|rrfa|user}

Specifies the type of test points to add to the scan chains.
Test points can be inserted after DFA or RRFA analysis or can be user-inserted.
By default, all test point types are inserted.

Example

The following command requests a preview of the scan chains after they would be updated with the test points inserted by the DFA analysis.

```
update_scan_chains -type dfa -preview \  
  -chains [find /des*/DLX_CORE -actual_scan_chains *]
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Inserting Scannable Test Points in Existing Scan Chains in Design for Test in Encounter RTL Compiler](#)

Affected by these commands: [insert_dft_dfa_test_points](#) on page 732

[insert_dft_rrfa_test_points](#) on page 754

[insert_dft_test_point](#) on page 768

[insert_dft_user_test_point](#) on page 773

Related command: [connect_scan_chains](#) on page 620

write_atpg

```
write_atpg
  { -cadence [-compression | > file]
  | -mentor [> file]
  | -stil [-dft_configuration_mode dft_config_mode]
    [> file]}
  [-decimals_ok] [-picoseconds]
  [-test_clock_waveform test_clock]
  [-apply_inputs_at integer]
  [-apply_bidirs_at integer]
  [-strobe_outputs_at integer]
  [-strobe_width integer] [design]
```

Writes out the scan-chain information for an Automatic Test Pattern Generator (ATPG) tool in a format readable by the designated ATPG tool.

The ATPG tool uses this information to generate appropriate test patterns. The file extension given to the interface file(s) is determined by the selected tool.

The interface file is useful only to the third-party tool if the test synthesis tool has connected the scan chain. Therefore, you should use this command only if the test synthesis tool has connected the scan chains.

Options and Arguments

-apply_bidirs_at *integer*

Specifies when in the test clock cycle to apply the bidirectional signals. Specify this time as a percentage of the test clock period.

Default: Same value as specified for **-apply_inputs_at**

-apply_inputs_at *integer*

Specifies when in the test clock cycle to apply the input signals. Specify this time as a percentage of the test clock period.

Default: 0

-cadence

Creates pin-assignment files that capture the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) for use by the Encounter Test ATPG tool.

If you use this option without the **-compression** option, the command writes out the pin-assignment information for full scan mode only. The information is written to the specified file.

Command Reference for Encounter RTL Compiler

Design for Test

-compression	Creates pin-assignment files for full scan mode, compression mode and XOR decompression mode. The following files are generated: <i>topmodulename</i> .FULLSCAN.pinassign, <i>topmodulename</i> .COMPRESSION.pinassign, and <i>topmodulename</i> .COMPRESSION_DECOMP.pinassign. Note: This option is only valid with the -cadence option.
-decimals_ok	Writes out decimal numbers. By default, time values are rounded off to integer numbers because many ATPG tools do not accept decimal numbers for test waveform time values. Use the -picoseconds option to minimize round-off errors.
<i>design</i>	Specifies the top module for which to write ATPG.
-dft_configuration_mode <i>dft_configuration_mode</i>	Writes ATPG information for the specified scan mode name. Note: This option is only valid with the -stil option.
<i>file</i>	Specifies the file to which the output must be written. If no file is specified, the output is written to standard out (stdout) and to the log file. Note: This argument is only valid with the -stil and -cadence options.
-mentor	Creates an interface file in the format used by the Mentor Graphics ATPG tool. Files generated: <ul style="list-style-type: none">■ <i>top_module</i>.testproc■ <i>top_module</i>.dofile
-picoseconds	Specifies to use picoseconds for the time unit. Use this option to minimize the round-off errors when rounding-off test waveform time values to integers. <i>Default:</i> nanoseconds
-stil	Creates an interface file in the IEEE Standard Test Interface Language (STIL) format (IEEE format 1450.1). Note: The generated STIL format is TetraMAX compatible.

Command Reference for Encounter RTL Compiler

Design for Test

`-strobe_outputs_at integer`

Specifies when in the test clock cycle to strobe the outputs.
Specify this time as a percentage of the test clock period.

Default: 40

`-strobe_width integer`

Specifies how long the outputs are valid during the test clock cycle. Specify this time as a percentage of the test clock period.

Default: 0

`-test_clock_waveform test_clock`

Specifies to use the clock waveform of the specified test clock.

Default: first test clock object found

Related Information

[Creating an Interface File for ATPG Tool in Design for Test in Encounter RTL Compiler](#)

Affected by this command: [compress_scan_chains](#) on page 601

[connect_scan_chains](#) on page 620

[define_dft_scan_clock_a](#) on page 680

[define_dft_scan_clock_b](#) on page 683

[define_dft_test_clock](#) on page 693

Affected by these attributes: [Actual Scan Chain Attributes](#)

write_bsdl

```
write_bsdl
  [-pinmap_file file]
  [-bsdl_package_name files]
  [-bsdlout file]
  [-include_private_instructions]
  [-expose_ports_with_pinmap]
  -directory string
```

Generates a file describing the boundary scan architecture of the design in Boundary Scan Description Language (BSDL), along with two VHDL package files, STD_1149_1_2001 and CDNDFT_1149_1_2001, which contain the supported boundary cell descriptions that were used during boundary scan insertion.

Note: Dedicated test-related signals (such as shift-enable, and test-mode signals defined without the -shared_in option) are also written to the BSDL file along with their respective compliance enable values.

Options and Arguments

-bsdl_package_name *files*

Specifies the name of a VHDL file or a comma-separated list of VHDL files, each containing one or more custom boundary cell descriptions that were used during boundary scan insertion.

The name of each package file is added to the BSDL file in a `use` statement.

Note: This option is required if you used custom boundary cells in the design.

-bsdlout *file*

Specifies the name of the BSDL file to be generated.

If you omit this option, the output file is named using the `topmodulename.bsdl`.

-directory *string*

Specifies the directory to which the output files must be written.

-expose_ports_with_pinmap

Specifies to only expose functional and test ports with package pinmap information to the output BSDL file.

Additionally, this option prevents the writing of other ports connected in the boundary-scan register without package pinmap information. The names of these other ports will not appear in the BOUNDARY_REGISTER section of the BSDL file. Rather, these port names will be represented with an asterisk (*) and their associated boundary-scan cells will be represented as INTERNAL as shown in the following BSDL snippet:

```
attribute BOUNDARY_REGISTER of test: entity is
  ...
  "8  (BC_OUT; *, INTERNAL, X)," &
```

When the -expose_ports_with_pinmap is not specified, all ports in the design will be written to each relevant section of the BSDL file, regardless of whether any pinmap information has been provided for any ports.

Package pinmap information may be provided during boundary-scan insertion or when writing the BSDL file using the -pinmap_file option.

-include_private_instructions

Specifies to include register access information for private instructions in the BSDL file.

-pinmap_file file Specifies the name of the pinmap file to be used to create a BSDL file.

Note: This file can have fewer pin-to-pad bonding requirements than the pinmap file specified for the boundary scan insertion.

Refer to [Pinmap File Format](#) for more information.

Example

- The following command creates a BSDL file named bsdlout. The name of the package file for the custom boundary cells is added to the BSDL file.

```
write_bsdl -directory . -bsdl_package_name MY_BIDIR_PKG -bsdlout bsdlout
```

This causes the following line to be added to file bsdlout:

```
use MY_BIDIR_PKG.all ;
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Writing a BSDL File in Design for Test in Encounter RTL Compiler](#)

Affected by this command: [insert_dft_boundary_scan](#) on page 717

Related constraints: [define_dft_shift_enable](#) on page 686

[define_dft_test_clock](#) on page 693

[define_dft_test_mode](#) on page 697

write_compression_macro

```
write_compression_macro
    {-chains integer | -scan_in integer -scan_out integer}
    -sub_chains integer
    [-decompressor {broadcast|xor }]
    [-compressor
        { xor [-scan_in_pipeline_depth integer]
            [-scan_out_pipeline_depth integer]
        | opmisr | hybrid
        | smartscan_xor -smartscan_ratio integer
            [-smartscan_no_update_stage] [-smartscan_serial_only]
            [-smartscan_pulse_width_multiplier {1|2|4}] }
        [-mask {wide0 | wide1 | wide2}]
        [-mask_sharing_ratio integer ]
        [-no_fullscan_muxing] [-jtag_control]
        [-serial_misr_read]
        [-shared_scan_in_pins string
            -asymmetrical_scan_in integer]
        [-block_select] [-compressed_chains integer]
        [-separate_mask_ports]
        [-remove_cancelling_terms_for_hierarchical_flow]
        [-low_pin_compression] [-bitwise_compressor]
        [-file file] [-info_file file]
```

Generates the RTL for a customized scan compression macro.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-asymmetric_scan_in integer

Specifies the number of scan in pins for an asymmetric compression macro.

This option is only supported with `wide1` and `wide2` types of channel masking.

For `wide1` type of masking, the number of scan data input pins must equal the number specified for the `-chains` option minus one (`number_of_chains-1`), while for `wide2` type of masking, the number must equal the specified number of chains minus two (`number_of_chains-2`).

Command Reference for Encounter RTL Compiler

Design for Test

-bitwise_compressor Generates a compressor with bitwise xorring of the channels (that is, each channel feeds only a single SO).

Use this option for leaf or mid-level blocks during hierarchical compression to avoid simulation issues.

-block_select Adds additional logic to bypass the block for compression inserted at the block level. In compression mode, the scan outputs are forced to zero. In uncompressed scan mode, the scan outputs are fed directly from the scan inputs.

An extra BLOCK_SELECT pin—to control the additional block select logic—is added to the compression macro.

If you specify this option with the -separate_mask_ports option, the mask registers are also bypassed by feeding the MASK_OUT ports directly from the MASK_IN ports.

Note: The BLOCK_SELECT pin should be held low to bypass the block and should be held high otherwise.

-chains integer Specifies the number of top level scan data input/scan data output pairs. Typically, this is the number of uncompressed scan chains. For MISR-based compression, the -chains option must be greater than or equal to 16.

You cannot specify this option when you specify the -scan_in and -scan_out options. When these options are specified, the number of chains equals the lesser of the following: the scan_in plus the number of shared control pins, or scan_out.

-compressed_chains integer

Specifies the number of scan chains in lower-level blocks that are already compressed.

The command adds the CCHAN_SI and CCHAN_SO ports to the macro to connect to the compressed chains at a lower block directly. The CCHAN_SO port feeds the channel data directly into the compressor.

This option cannot be specified when the -compressor option is set to smartscan_xor.

-compressor {xor | opmisr | hybrid | smartscan_xor}

Specifies the type of compression logic to be built:

- xor specifies to build an XOR-based compressor

Command Reference for Encounter RTL Compiler

Design for Test

- `opmisr` specifies to build a MISR-based compressor
- `hybrid` specifies to build MISR compression with MISR bypass capability to effectively result in an XOR-based compressor.
- `smartscan_xor` specifies to add smartscan logic to the XOR-based compression macro.

Default: xor

`-decompressor {broadcast | xor}`

Specifies the type of decompression logic to be built:

- `xor` specifies to build an XOR-based spreader network in addition to the broadcast-based decompression logic
- `broadcast` specifies to build a broadcast-based decompression logic (simple scan fanout).

`-file file`

Specifies the filename where the compression macro RTL will be written. If not specified, the RTL will be written to `stdout`.

`-info_file file`

Specifies a file containing more detailed information about the compression macro. This script can be sourced into the current session to provide more information about the compression macro to commands such as `write_et` so they can generate accurate input files for Encounter Test.

`-jtag_control`

Specifies to include a JTAG-controlled test data register (TDR) which generates compression test signals to configure the compression testmode.

`-low_pin_compression`

Enables low pin count compression.

`-mask {wide0 | wide1 | wide2}`

Inserts scan channel masking logic of the specified type.

The masking types that can be used depend on the compressor type specified with the `-compressor` option.

`-mask_sharing_ratio integer`

Specifies the number of internal scan channels sharing a mask register. The specified integer may not exceed the value specified for the compression ratio.

Note: This option is only valid with `wide1` and `wide2` masking.

Command Reference for Encounter RTL Compiler

Design for Test

`-no_fullscan_muxing` Specifies to exclude additional muxing logic to the compression macro. By default, additional muxing is added to the compression macro to concatenate the compressed scan channels into uncompressed fullscan chains. If such muxing exists outside the compression macro, specify this option to exclude this logic from the compression macro.

`-remove_cancelling_terms_for_hierarchical_flow`

Generates a macro which prevents cancellation of compression channels within higher-level compression macros. Each scan output is fed by the XOR of an odd number of channels.

`-scan_in integer, -scan_out integer`

Specify the width of the `RSI_SI` and `DSO_SO` ports (the number of scan data inputs and scan data outputs to the compression macro) respectively.

These options apply for MxN compression in any of the following cases:

- $M < N$
- $M < N-1$ and wide1 masking is used: one of the scan data inputs can be shared with the CME pin
- $M < N-2$ and wide2 masking is used: two of the scan data inputs can be shared with the CME0 and CME1 pins
- $M > N$

`-scan_in_pipeline_depth integer`

Specifies the number of pipeline stages required at the scan data input side.

`-scan_out_pipeline_depth integer`

Specifies the number of pipeline stages required at the scan data output side.

`-separate_mask_ports`

Creates separate `MASK_IN` and `MASK_OUT` ports that are used for block level compression processing in the hierarchical compression flow.

The number of `MASK_IN/MASK_OUT` ports that is added, is the same number as the number of scan data input ports.

Command Reference for Encounter RTL Compiler

Design for Test

-serial_misr_read Specifies to include support for reading MISR bits serially through the scan data pins.

-shared_scan_in_pins *string*

Specifies the pins that can be shared with the scan data input pins. Separate the pin names using one or more blanks.

- For -mask wide1, specify *CME*
- For -mask wide2, you can specify the following values:
CME0, *CME1*, or *CME0 CME1*

This option enables the use of an asymmetrical compression macro.

This option will be ignored and a warning will be given if you also specified the **-scan_in** and **-scan_out** options and the value for **scan_in** is larger than the value for **scan_out**.

-smartscan_no_update_stage

Prevents the insertion of update registers between the deserializer and the decompressor. In this case, lockup latches are inserted between the deserializer flops and the decompressor.

You cannot specify this option when you have set the **-smartscan_pulse_width_multiplier** option to either 2 or 4 .

By default, the tool inserts update registers between the deserializer and the decompressor.

-smartscan_pulse_width_multiplier {1|2|4}

Determines whether to add clock divider logic to widen the clock pulse going to the scan chains. You can specify the following values:

- 1—no logic added
- 2—increases the scan path through the SmartScan clock controller with 1 bit
- 4—increases the scan path through the SmartScan clock controller with 2 bits

Default: 1

Command Reference for Encounter RTL Compiler

Design for Test

`-smartscan_ratio integer`

Specifies the number of parallel scan data input pins that correspond to a single serial scan data input pin. The number of defined (fullscan) chains must be an integral multiple of the specified smartscan ratio.

`-smartscan_serial_only`

Specifies to only insert the smartscan serial-only interface. The number of deserializer (and serializer) registers will match the number of the defined chains. When building the model for Encounter Test you will need to add the pseudo pins for the parallel interface.

`-sub_chains integer`

Specifies the number of compressed scan channels that exist in the design or that you will build.

Examples

- The following command writes an XOR-based compression macro without masking to the file xor1.v.

```
rc:/> write_compression_macro -compressor xor -chains 8 -sub_chains 88 \
    -file xor1.v
Checking out license 'Encounter_Test_Architect'... (1 seconds elapsed)
...
```

- The following command writes an XOR-based compression macro with masking logic of type wide1. The `-no_fullscan_muxing` option is specified so the logic to concatenate the compressed `sub_chains` into fullscan chains will be excluded.

Note: Since the `-no_fullscan_muxing` option is specified, the number of `sub_chains` is no longer required to be evenly divisible by the number of chains.

```
rc:/> write_compression_macro -compressor xor -mask wide1 -chains 8 \
    -sub_chains 85 -no_fullscan_muxing -file xor2.v
Checking out license 'Encounter_Test_Architect'... (1 seconds elapsed)....
```

- The following command writes an MISR-based compression macro with masking logic of type wide1, with decompression logic of type xor. The compressor type is hybrid which means the MISR can be bypassed resulting in XOR compression.

```
rc:/> write_compression_macro -compressor hybrid -decompressor xor \
    -mask wide1 -chains 16 -sub_chains 512 -file hybrid1.v
Checking out license 'Encounter_Test_Architect'... (2 seconds elapsed)
...
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following command writes a MISR-based compression macro with masking logic of type wide2. The `-info_file` option is also specified.

```
rc:/> write_compression_macro -compressor opmisr -mask wide2 -chains 20 \
    -sub_chains 500 -file opmisrl.v -info_file opmisrl.info
....
```

- The following command generates an asymmetric pipelined XOR-based compression macro, with masking logic of type wide2, with five top-level chains, of which two are shared with the mask control signals, with separate mask ports, with additional logic to bypass the block for compression inserted at the block level, and with two pipeline stages at the scan data output side.

```
write_compression_macro -chains 5 -sub_chains 9 -asymmetric_scan_in 3 \
    -shared_scan_in_pins "CME0 CME1" -mask wide2 -separate_mask_ports \
    -block_select -scan_out_pipeline_depth 2 -file my_comp_macro.v
```

- The following command generates a pipelined XOR-based compression macro, with masking logic of type wide1, with two top-level chains, with one pipeline stage at the scan data input side, six pipeline stages at the scan data output side, and without additional muxing logic.

```
write_compression_macro -chains 2 -sub_chains 5 -mask wide1 \
    -scan_in_pipeline_depth 1 -scan_out_pipeline_depth 6 -no_fullscan_muxing
```

- The following command generates an asymmetric Smartscan-based compression macro with both parallel and serial interface, with masking logic of type wide1, with eight fullscan chains, 32 compression channels and a Smartscan ratio of 4.

```
write_compression_macro -chains 8 -sub_chains 32 -mask wide1 \
    -compressor smartscan_xor -decompressor xor -smartscan_ratio 4
```

- The following command generates an asymmetric Smartscan-based compression macro with serial interface only, with masking logic of type wide1, with two fullscan chains, 32 compression channels and a Smartscan ratio of 4.

```
write_compression_macro -chains 2 -sub_chains 32 -mask wide1 \
    -compressor smartscan_xor -decompressor xor -smartscan_ratio 4 \
    -smartscan_serial_only
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Low Pin Count Compression Using Encoded Compression Signals](#)
- [Manually Inserting a Scan Compression Macro](#)
- [Using Asymmetrical Scan Compression](#)
- [MxN Compression](#)

write dft abstract model

```
write_dft_abstract_model [-ctl]
    [-include_compression_information]
    [-dft_configuration_mode dft_config_mode]
    [design] [> file]
```

Writes a scan abstract model for all the top-level scan chains configured in the design.

Note: Currently, this command is not supported for the clocked LSSD scan style with the -ctl option.

Options and Arguments

-ctl Writes out a scan abstract model in the Core Test Language (CTL) format (IEEE format P1450.6).

If you omit this option, the scan abstract model is written in native RC format that consists of a list of `define_dft_abstract_segment` commands, one per top-level scan chain.

design Specifies the design for which to write out the scan abstract model of the scan chains.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-dft configuration mode *dft configuration mode*

Writes scan chain information related to the specified scan mode name.

file Specifies the file to which the output must be written.

You can write out the CTL file in compressed format by specifying a file name with the .gz extension.

Default: output is written to the screen

- include compression information

Adds the required compression information created by the `compress_block_level_chains` command. This information is used at the next level of integration in the hierarchical compression flow.

Command Reference for Encounter RTL Compiler

Design for Test

Examples

- In the following example, the different active edges of the different test clocks in the same test clock domain are allowed to be mixed on the same scan chains. Following shows the configuration result and the scan abstract models for the scan chains:

```
rc:> connect_scan_chains
      Configuring 1 chains for 27 scan f/f
      Configured 1 chains for Domain: 'clkAll', edge: 'mixed'
      AutoChain_1 (DFT_sdi_1 -> DFT_sdo_1) has 27 registers; Domain:clkAll,
      edge: mixed
      Processing 1 scan chains in 'muxed scan' style.
      Using default shift enable signal 'SE': '/designs/test/ports_in/SE' active
      high.
      Connecting scan chain 'AutoChain_1' with 27 flip-flops.
      Mapping DFT logic introduced by scan_chain connection...
      Mapping DFT logic done.
      Reporting 1 scan chain

Chain 1: AutoChain_1
  scan_in:      DFT_sdi_1
  scan_out:     DFT_sdo_1
  shift_enable: SE (active high)
  clock_domain: clkAll (edge: mixed)
  length: 27
    bit 1       out1_reg_4 <test_clk1/fall>
    ...
    bit 5       out1_reg_8 <test_clk1/fall>
    llatch 5   DFT_Lockup_g1
    bit 6       out2_reg_4 <test_clk2/fall>
    ...
    bit 10      out2_reg_8 <test_clk2/fall>
    llatch 10  DFT_Lockup_g348
    bit 11      out3_reg_4 <test_clk3/fall>
    ...
    bit 18      out1_reg_2 <test_clk1/rise>
    bit 19      out1_reg_3 <test_clk1/rise>
    llatch 19  DFT_Lockup_g349
    bit 20      out2_reg_0 <test_clk2/rise>
    ...
    bit 23      out2_reg_3 <test_clk2/rise>
    llatch 23  DFT_Lockup_g350
    bit 24      out3_reg_0 <test_clk3/rise>
    ...
    bit 27      out3_reg_3 <test_clk3/rise>
-----
rc:/> write_dft_abstract_model
scan style is muxed_scan

# writing abstract model for 1 scan chain

define_dft abstract_segment -module test \
  -name test AutoChain_1 \
  -sdi DFT_sdi_1 -sdo DFT_sdo_1 \
  -shift_enable_port SE -active high \
  -clock_port clk1 -fall \
  -tail_clock_port clk3 -tail_edge_rise \
  -length 27
```

Command Reference for Encounter RTL Compiler

Design for Test

To avoid naming collisions when reading in a scan abstract model, the segment names are prefixed with the module name.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Creating a Scan Abstract Model](#)
- [Hierarchical Compression Flow](#)
- [Bottom-Up Test Synthesis Flow](#)

Affected by these commands: [compress_block_level_chains](#) on page 598

[connect_scan_chains](#) on page 620

Affected by these attributes: [Actual Scan Chain](#) attributes

[dft_include_controllable_pins_in_abstract_model](#)

Related command: [read_dft_abstract_model](#) on page 783

[write_hdl](#) on page 255 (-abstract)

write_dft_rtl_model

```
write_dft_rtl_model  
  -directory directory  
  [design]
```

Writes out an RTL model of the design in Verilog if the DFT RTL insertion update flow is enabled. This command minimally modifies the user-supplied RTL files to include the RTL constructs of the inserted JTAG macro and MBIST structures.

Options and Arguments

<i>design</i>	Specifies the design for which to update the RTL files. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
<i>-directory directory</i>	Specifies the directory to which the updated RTL files must be written. The directory is created if it does not exist.

Related Information

[MBIST Top-Down RTL Insertion Flow in Design for Test in Encounter RTL Compiler](#)

Affected by these commands [insert_dft_jtag_macro on page 736](#)

[insert_dft_mbist on page 743](#)

Affected by this attribute: [dft_rtl_insertion](#)

write_et_atpg

```
write_et_atpg
  [-ncsim_library string [-library string]]
  [-directory string] [-run_from_et_workdir]
  [ -configuration_mode_order dft_configuration_mode_list]
  [ -delay ]
  { [-opcg_mode opcg_mode]
  | [-compression]
    [-dft_configuration_mode dft_config_mode]...
    [-build_faultmodel_options string]
    [-build_model_options string]
    [-build_testmode_options string]
    [-atpg_options string] [-effort string]
    [-verify_test_structures_options string] }
  [-force] [design]
```

Writes out the necessary files and the template run scripts to run Automatic Test Pattern Generator (ATPG) using the Encounter Test software.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

The generated scripts depend on the setting of the `dft_true_time_flow` root attribute.

- With the default setting of the attribute, the command generates run scripts for static ATPG and non-SDF based OPCG delay test flows.
- When you enable the true time flow by setting this attribute to `true`, the command will also generate the `tt_setup` file. The Encounter Test `true_time` command is used to generate the ATPG script. This option provides scripts for the following ATPG test flows:

static ATPG
OPCG Delay Test
non-OPCG Delay Test
SDF/SDC based Delay Test (OPCG or non-OPCG)
RAM Sequential Delay Test
Path Delay Test
Iddq Test

For the last four flows, you will need to make minor modifications to the `tt_setup` file written by `write_et_atpg`, and you will need to regenerate the scripts using the Encounter Test `true_time` command. For more information on the `true_time` script and `tt_setup` file, refer to the [Encounter Test Automatic Test Pattern Generation User Guide](#).

Note: In the true timing flow, several options are not applicable, while the behavior of

Command Reference for Encounter RTL Compiler

Design for Test

other options differs. See the option descriptions for more information.

By default, this command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis in an assumed scan mode.

Note: Because the true time flow does not support the assumed scan mode, this file is not written out in the true time flow.

- `et.modedef`—A mode definition file, a text file that describes the test mode in assumed scan mode
- `topmodulename.ASSUMED.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design

Note: Because the true time flow does not support the assumed scan mode, this file is not written out in the true time flow.

- `topmodulename.FULLSCAN.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- `runet.atpg`—A template script file to run the requested testability analysis
- `topmodulename.et_netlist.v`—A netlist for Encounter Test
- If the `write_et_atpg` command is run with the `-compression` option, the following pin-assignment files are generated in addition to the `topmodulename.FULLSCAN.pinassign` file. In this case, all three files include the compression test signals with their appropriate test functions to validate their specific test mode:
 - `topmodulename.COMPRESSION_DECOMP.pinassign`—A file generated *only* when inserting XOR-based decompression logic
 - `topmodulename.COMPRESSION.pinassign`—A file generated when inserting broadcast-based decompression logic
- If the `write_et_atpg` is specified with the `-delay` option, the following files are generated in addition to the `topmodulename.FULLSCAN.pinassign` file:
 - `topmodulename.FULLSCAN_OPCGModeName.pinassign`—A pin assignment file generated only when inserting OPCG logic in full scan mode

This file includes the OPCG test signals with their appropriate test functions and includes the oscillator and domain related information.

Command Reference for Encounter RTL Compiler

Design for Test

- ❑ *topmodulename.FULLSCAN_OPCGModeName.seqdef*—A sequence definition file generated when inserting OPCG logic in full scan mode.

This file is used to initialize the PLL and establish the OPCG mode.

All of the files are used to validate their specific test mode.

- If the `write_et_atpg` is specified with the `-compression` and `-delay` options, the following files are generated in addition to the

topmodulename.FULLSCAN.pinassign and

topmodulename.COMPRESSION.pinassign files.

- ❑ *topmodulename.COMPRESSION_OPCGModeName.pinassign*—A pin assignment file generated *only* when inserting OPCG logic with XOR-based decompression logic

This file includes the OPCG test signals with their appropriate test functions and includes the oscillator and domain related information.

- ❑ *topmodulename.COMPRESSION_OPCGModeName.seqdef*—A sequence definition file generated when inserting OPCG logic with XOR-based compression logic

This file is used to initialize the PLL and establish the OPCG mode.

All of the files are used to validate their specific test mode.

- `run_compression_decomp_sim`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test for compression logic built using XOR-based decompression logic

- `run_compression_sim`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test for compression logic built using broadcast-based decompression logic

- `run_fullscan_sim`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test in full scan mode.

- `run_fullscan_sim_OPCGModeName`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test in full scan mode with OPCG logic

- `run_compression_sim_OPCGModeName`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test for compression logic built using XOR-based decompression logic and with OPCG logic.

Note: Some files can be customized according to the setup requirements.

Command Reference for Encounter RTL Compiler

Design for Test

In the true flow, the command writes out all files related to all configuration modes in the design.

- If the `write_et_atpg` command is specified with the `-opcg_mode` option, the following files are written out in addition:
 - ❑ `topModuleName.FULLSCAN_OP CGModeName.pinassign`
 - ❑ `topModuleName.FULLSCAN_OP CGModeName.seqdef`
- If you also specified the `-compression` option, the following files are also written:
 - ❑ `topmodulenname.COMPRESSION_OP CGModeName.pinassign`
 - ❑ `topmodulenname.COMPRESSION_OP CGModeName.seqdef`

Options and Arguments

`-atpg_options {option1=value option2=value}`

Specifies extra ATPG analysis options.

Note: This option does not apply to the true time flow.

`-build_faultmodel_options string`

Specifies a string containing the extra options to build a fault model.

Note: For more information on these options, refer to the `build_faultmodel` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option does not apply to the true time flow.

`-build_model_options {option1=value option2=value}`

Specifies extra options to apply when building the Encounter Test model.

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option does not apply to the true time flow.

`-build_testmode_options {option1=value option2=value}`

Specifies extra options to apply when building the test mode for Encounter Test.

Command Reference for Encounter RTL Compiler

Design for Test

Note: For more information on these options, refer to the `build_testmode` command in the *Command Line Reference* (of the Encounter Test documentation).

Note: This option does not apply to the true time flow.

`-compression`

Instructs to write out the files needed to run ATPG-based testability analysis for compression mode.

`-configuration_mode_order dft_configuration_mode...`

Specifies to write Encounter Test script files for a compression mode. Valid compression mode names are:

`COMPRESSION, COMRESSION_DECOMP, OPMISRPLUS,
OPMISRPLUS_DECOMP, FULLSCAN`

Note: If specified, the `FULLSCAN` compression mode must be specified last.

- In the default flow, you cannot combine this option with the `-delay` option.
- In the true time flow, you can combine this option with the `-delay` option, but you can only specify two configuration modes because the true time flow can only handle two modes.

`-delay`

Specifies to generate additional files for Encounter Test to verify the OPCG logic.

- In the default flow, you cannot combine this option with the `-configuration_mode_order` option.
- In the true time flow, you can combine this option with the `-configuration_mode_order` option. In this flow, this option specifies to write out delay tests even when no OPCG logic was inserted.

`design`

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

Command Reference for Encounter RTL Compiler

Design for Test

`-dft_configuration_mode dft_configuration_mode...`

Writes scan chain information related to the specified scan mode name(s).

Note: This option does not apply to the true time flow.

`-directory string`

Specifies the directory to which the output files must be written.

Default: `current_working_directory/et_scripts`

`-effort {low | medium | high}`

Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more.

Default: low

Note: This option does not apply to the true time flow.

`-force`

Specifies to continue even when the DFT logic appears to be incomplete.

`-library string`

Specifies the list of Verilog structural library files required to run Encounter Test ATPG.

You can specify the files explicitly or you can specify an *include* file that lists the files. You can also specify directories of Verilog files but you cannot reference directories in an include file.

For example, if the Verilog files required to run ATPG are:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_et_atpg` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_atpg -library "./padcells.v ./stdcells.v \  
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

Command Reference for Encounter RTL Compiler

Design for Test

And then specify the following:

```
write_et_atpg -library "include_libraries.v ./
memories\"\
./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

-ncsim_library *string*

Specifies the list of library files required for the Incisive Enterprise simulation of the generated vectors.

For more information on how to specify the list of files, refer to the **-library** option.

-opcg_mode *opcg_mode*

Specifies the OPCG mode for which the delay tests must be generated.

Note: This option only applies to the true time flow.

-run_from_et_workdir

Allows to run the Automatic Test Pattern Generator (ATPG) from the working directory specified with the **-directory** option.

Default: parent directory of the specified work directory.

-verify_test_structures_options {*option1=value* *option2=value*}

Specifies extra options to apply when performing test structure verification for Encounter Test.

Note: This option does not apply to the true time flow.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

```
write_et_atpg -directory atpg -library $sim/mylib.v
Examining the atpg directory that was generated shows the following files:
rc:/> shell ls atpg
run_compression_decomp_sim
run_compression_sim
runet.atpg
run_fullscan_sim
test.COMPRESSION_DECOMP.pinassign
test.COMPRESSION.pinassign
test.et_netlist.v
test.FULLSCAN.pinassign
test.rc_netlist.v
```

Command Reference for Encounter RTL Compiler

Design for Test

- The following command uses the `-configuration_mode_order` option to generate the files to run an ATPG-based testability analysis first using the `OPMISRPLUS_DECOMP` compression mode and then with the `FULLSCAN` mode.

```
write_et -atpg -configuration_mode_order {OPMISRPLUS_DECOMP FULLSCAN} \
          -directory rc_et
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Using Encounter Test to Analyze Testability](#)
- [Writing the Scripts and Setup Files to Perform ATPG](#) in “Exporting the Design”
- [Defining the Oscillator Sources](#) in “Inserting On-Product Clock Generation Logic”
- [Generate Files for ATPG and Simulation](#) in “Inserting On-Product Clock Generation Logic”
- [Compression Logic Verification and Test Vector Generation Using Encounter Test](#) in “Inserting Scan Chain Compression Logic”
- [Using Encounter Test to Perform a Deterministic Fault Analysis on a Scan Connected Netlist](#)

write_et_bsv

```
write_et_bsv -library string
    [-bsdl file [-bsdl_package_path string]
     [-bsdl_package_name files]]
    [-build_model_options string]
    [-directory string] [-run_from_et_workdir]
    [design]
```

Writes out the necessary files and the template run scripts to run boundary scan verification.

This command generates the following files:

- *topmodulename.bsdl*—A BSDL file
- *topmodulename.et_netlist.v*—A netlist for Encounter Test
- *runet.bsv*—An Encounter Test run file to run Boundary Scan Verification (BSV).

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

-bsdl *file*

Specifies the name of the BSDL file to be used for the boundary scan verification.

If you omit this option but you specified the **-bsv** option, this command will automatically run the **write_bsdl** command to generate the BSDL file.

Important

You must use this option if you inserted custom boundary cells in the design. Additionally, the BSDL file should be written using the **write_bsdl** command with the **-bsdl_package_name** option to list the custom package file to be used during boundary scan verification.

Command Reference for Encounter RTL Compiler

Design for Test

`-bsdl_package_name files`

Specifies a package file or a comma-separated list of package files that describe the custom boundary cells used in the design.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`bsdl_package_path string`

Specifies the UNIX directory or a comma-separated list of directories that indicate(s) where to find the package file(s).

You can use dot (.) to refer to the current working directory.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-build_model_options {option1=value option2=value}`

Specifies extra options to apply when building the Encounter Test model.

`design`

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory string`

Specifies the directory to which the output files must be written.

Default: current_working_directory/et_scripts

`-library string`

Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and Incisive Enterprise simulation of the generated vectors must be provided using the `-library` option of the `write_et_bsv` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_bsv` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and Incisive Enterprise simulation are:

Command Reference for Encounter RTL Compiler

Design for Test

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_et_bsv` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_bsv -library "./padcells.v ./stdcells.v \  
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

And then specify the following:

```
write_et_bsv \  
-library "include_libraries.v ./memories \  
./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-run_from_et_workdir`

Allows to run boundary scan verification from the working directory specified with the `-directory` option.

Default: parent directory of the specified work directory.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

```
write_et_bsv -directory bsv -library $sim/mylib.v
```

Examining the `atpg` directory that was generated shows the following files:

```
rc:/> shell ls bsv  
topmodulename.bsdl  
runet.bsv  
test.et_netlist.v  
test.rc_netlist.v
```

Related Information

[Generating Script for Boundary Scan Verification in Design for Test in Encounter RTL Compiler](#)

write_et_dfa

```
write_et_dfa
  [-library string]
  [-effort string] [-build_model_options string]
  [-build_testmode_options string]
  [-atpg_options string] [-dfa_options string]
  [-include_redundant_faults]
  [-verify_test_structures_options string]
  [-directory string] [-run_from_et_workdir]
  [design]
```

Writes out the necessary files and the template run scripts to run Deterministic Fault Analysis using the Encounter Test software. The template script will only be written if actual scan chains exist in the design. DFA analysis is not supported in assumed scan mode.

This command generates the following files:

- *topmodulename.FULLSCAN.pinassign*—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- *runet.dfa* —A template script file to run the requested deterministic fault analysis
- *topmodulename.et_netlist.v*—A netlist for Encounter Test
- *TestPointInsertion.testmode_name.dfa*—A file containing test point locations.
- *TestPointInsertion.FULLSCAN_inactive.dfa*—A file containing the additional test point locations found when the inactive faults are included during DFA analysis
- *run_fullscan_sim*—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test in full scan mode.

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to [Encounter Test Product Requirements for Advanced Features](#) in *Design for Test in Encounter RTL Compiler*.

Options and Arguments

-atpg_options {option1=value option2=value}

Specifies extra ATPG analysis options.

-build_model_options {option1=value option2=value}

Specifies extra options to apply when building the Encounter Test model.

-build_testmode_options {option1=value option2=value}

Specifies extra options to apply when building the test mode for Encounter Test.

design

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-dfa_options {option1=value option2=value}

Specifies extra options for deterministic fault analysis.

-directory string Specifies the directory to which the output files must be written.

Default: current_working_directory/et_scripts

-effort {low | medium | high}

Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more.

Default: low

-include_redundant_faults

Specifies to include the redundant faults during DFA analysis.

Command Reference for Encounter RTL Compiler

Design for Test

-library *string* Specifies the list of Verilog structural library files.

The Verilog libraries required to run Encounter Test ATPG and Incisive Enterprise simulation of the generated vectors must be provided using the **-library** option of the `write_et_dfa` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_dfa` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and Incisive Enterprise simulation are:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_et_dfa` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_dfa -library "./padcells.v ./stdcells.v \  
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

And then specify the following:

```
write_et_dfa \  
-library "include_libraries.v ./memories \  
./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

-run_from_et_workdir

Allows to run the Deterministic Fault Analysis from the working directory specified with the **-directory** option.

Default: parent directory of the specified work directory.

-verify_test_structures_options {option1=value option2=value}

Specifies extra options to apply when performing test structure verification for Encounter Test.

Examples

- The following command generates the files to run deterministic fault analysis.

```
write_et_dfa -directory dfa -library $sim/mylib.v
```

Examining the `dfa` directory that was generated shows the following files:

```
rc:/> shell ls dfa
runet.dfa
run_fullscan_sim
test.et_netlist.v
test.FULLSCAN.pinassign
test.rc_netlist.v
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Exporting the Design for Test Point Selection](#)
- [Using Encounter Test to Perform a Deterministic Fault Analysis on a Scan Connected Netlist](#)

Related command: [insert_dft_dfa_test_points](#) on page 732

write_et_lbist

```
write_et_lbist
  [-library string] [-directory string]
  [-build_model_options {option1=value option2=value}]
  [-build_testmode_options {option1=value option2=value}]
  [-build_faultmodel_options {option1=value option2=value}]
  [-verify_test_structures_options {option1=value option2=value}]
  [-run_from_et_workdir] [design]
```

Writes out data and script files for Encounter Test to perform Logic Built-in Self Test.

This command generates the following files:

- *topmoduleName.et_netlist.v*—A netlist for Encounter Test
- *assignfile.JTAG.instructionName*—A pin-assignment file for the parent mode for RUNBIST/SETBIST instruction
- *assignfile.LBIST.instructionName*—A pin-assignment file for the child mode for RUNBIST/SETBIST instruction
- *MODE_JTAG_instructionName*—A mode definition file that describes the testmode in parent mode for RUNBIST/SETBIST instruction
- *MODE_LBIST*—A mode definition file that describes the testmode in child mode for LBIST
- *TBDseqpatt.JTAG_instructionName*—A sequence definition file for parent mode for RUNBIST/SETBIST instruction
- *TBDseqpatt.LBIST_instructionName*—A sequence definition file for child mode for RUNBIST/SETBIST instruction
- *TestSequence.seq*—Test sequence file for LBIST test
- *run_lbist_instructionName*—An Encounter Test file to run LBIST tests for RUNBIST/SETBIST instruction
- *run_sim_instructionName*—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test for LBIST.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-build_faultmodel_options {option1=value option2=value ...}`

Specifies a string containing the extra options to build a fault model.

Note: For more information on these options, refer to the `build_faultmodel` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_model_options {option1=value option2=value ...}`

Specifies extra options to apply when building the Encounter Test model.

Note: For more information on these options, refer to the `build_model` command in the *Command Line Reference* (of the Encounter Test documentation).

`-build_testmode_options {option1=value option2=value ...}`

Specifies extra options to apply when building the test mode for Encounter Test.

Note: For more information on these options, refer to the `build_testmode` command in the *Command Line Reference* (of the Encounter Test documentation).

`design`

Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory string`

Specifies the directory to which the output files must be written.

Default: current_working_directory/et_scripts

`-library string`

Specifies the list of Verilog structural library files required to run Encounter Test.

You can specify the files explicitly or you can specify an *include* file that lists the files. You can also specify directories of Verilog files but you cannot reference directories in an include file.

For example, if the following Verilog files are required:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_et_lbist` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_lbist -library "./padcells.v ./stdcells.v \
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"
'include "./stdcells.v"
```

And then specify the following:

```
write_et_lbist -library "include_libraries.v ./
memories \
./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-run_from_et_workdir`

Allows to perform Logic Built-in Self Test from the working directory specified with the `-directory` option.

Default: parent directory of the specified work directory.

`-verify_test_structures_options {option1=value option2=value}`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

- [Inserting LBIST Logic](#)
- [Generating Files for LBIST Pattern Generation and Simulation](#)

Affected by these commands: [insert_dft logic_bist](#) on page 741

[write_logic_bist_macro](#) on page 850

write_et_mbist

```
write_et_mbist
    -mbist_interface_file_dir string
    -mbist_interface_file_list string
    [-build_model_options string]
    [-create_embedded_test_options string]
    [-bsv [-bsdl file [-bsdl_package_path string]
           [-bsdl_package_name files] ] [-library string]
     [-directory string] [-run_from_et_workdir]
     [-force] [design]
```

Writes out the necessary files and the template run scripts to run Create Embedded Test using the Encounter Test software.

This command generates the following files:

- *topmodulename.ASSUMED.pinassign*—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- *topmodulename.FULLSCAN.pinassign*—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- *topmodulename.bsdl*—A BSDL file produced when specifying the `-bsv` and BSDL-related options.
- *runet.mbist*—An Encounter Test run file to run Boundary Scan Verification (BSV) when specifying the `-bsv` option.
- *topmodulename.et_netlist.v*—A netlist for Encounter Test
- *runet.mbist_interface*—A template script file to run Create Embedded Test in Encounter Test

Note: Some files can be customized according to the setup requirements.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

Options and Arguments

`-bsdl file` Specifies the name of the BSDL file to be used for the boundary scan verification.

Command Reference for Encounter RTL Compiler

Design for Test

If you omit this option but you specified the `-bsv` option, this command will automatically run the `write_bsdl` command to generate the BSDL file.

Important

You must use this option if you inserted custom boundary cells in the design. Additionally, the BSDL file should be written using the `write_bsdl` command with the `-bsdl_package_name` option to list the custom package file to be used during boundary scan verification.

`-bsdl_package_name files`

Specifies a package file or a comma-separated list of package files that describe the custom boundary cells used in the design.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-bsdl_package_path string`

Specifies the UNIX directory or a comma-separated list of directories that indicate(s) where to find the package file(s).

You can use dot (.) to refer to the current working directory.

Note: This option is only required if you used custom boundary cells in the design. This option cannot be specified without the `-bsdl` option.

`-bsv`

Writes out the files needed for boundary scan verification.

Note: This option prints a pin assignment file if differential PAD pairs are detected in the design.

`-build_model_options {option1=value option2=value}`

Specifies extra options to apply when building the Encounter Test model.

`-create_embedded_test_options {option1=value option2=value}`

Specifies extra options to apply when running Create Embedded Test in Encounter Test.

Command Reference for Encounter RTL Compiler

Design for Test

<i>design</i>	Specifies the design for which to write out the Encounter Test input files. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
-directory <i>string</i>	Specifies the directory to which the output files must be written. <i>Default:</i> <i>current_working_directory/et_scripts</i>
-force	Writes out the scripts when MBIST was inserted using direct access mode. Normally this command requires a JTAG macro to generate the scripts. If you insert MBIST using the direct access mode, you must use the -force option to successfully complete the command.
-library <i>string</i>	Specifies the list of Verilog structural library files. The Verilog libraries required to run Encounter Test ATPG and Incisive Enterprise simulation of the generated vectors must be provided using the -library option of the <code>write_et_mbist</code> command. These libraries must be in a structural format. The files can be specified separately on the <code>write_et_mbist</code> command line or can be referenced using an <i>include</i> file. Directories of Verilog files can also be specified but they cannot be referenced in the include file. For example, if the Verilog files required to run ATPG and Incisive Enterprise simulation are:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_et_mbist` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_mbist -library "./padcells.v ./stdcells.v \  
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

And then specify the following:

```
write_et_mbist -library "include_libraries.v \  
./memories ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

-mbist_interface_file_dir

Specifies the MBIST interface file directories. Separate the directory names with blank spaces.

-mbist_interface_file_list

Specifies a list of MBIST interface files. Separate the file names with commas, for example, file1,file2.

-run_from_et_workdir

Allows to run Create Embedded Test from the working directory specified with the **-directory** option.

Default: parent directory of the specified work directory.

Examples

- The following command generates the files to run Boundary Scan Verification and Create Embedded Test in Encounter Test.

```
write_et_mbist -mbist_interface_file_dir directory \
-mbist_interface_file_list file1,file2 -bsv -library $sim/mylib.v
```

Examining the atpg directory that was generated shows the following files:

```
rc:/> shell ls mbist
test.COMPRESSION_DECOMP.pinassign
test.COMPRESSION.pinassign
test.et_netlist.v
test.FULLSCAN.pinassign
test.rc_netlist.v
runet.mbist
runet.mbist_interface
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*:

[Writing the Scripts and Setup Files to Generate MBIST Patterns](#)

[MBIST Top-Down Gate Insertion Flow](#)

write_et_rrfa

```
write_et_rrfa
    [-atpg] [-force] [-effort string] [-atpg_options string]
    [-rrfa_options string] [-build_model_options string]
    [-build_testmode_options string]
    [-verify_test_structures_options string] [-library string]
    [-directory string] [-run_from_et_workdir] [design]
```

Writes out the necessary files and the template run scripts to run either Automatic Test Pattern Generator (ATPG) or Random Resistance Fault Analysis (RRFA) based testability analysis, generate test patterns using the Encounter Test software.

This command generates the following files:

- `et.exclude`—A file listing objects to be excluded from the ATPG analysis in an assumed scan mode.
- `et.modedef`—A mode definition file, a text file that describes the test mode in assumed scan mode
- `topmodulename.ASSUMED.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock) and their test function used to build the testmode before actual scan chains exist in the design
- `topmodulename.FULLSCAN.pinassign`—A pin-assignment file that captures the top-level scan-related signals (shift-enable, test-mode, test-clock and scan data IOs) and their test function used to build the testmode when actual scan chains exist in the design
- `run_fullscan_sim`—An Incisive Enterprise simulator run file used to simulate the test patterns created by Encounter Test in full scan mode.

Note: Some files can be customized according to the setup requirements.

For more information on the exact Encounter Test product requirements, refer to [Encounter Test Product Requirements for Advanced Features](#) in *Design for Test in Encounter RTL Compiler*.

Command Reference for Encounter RTL Compiler

Design for Test

Options and Arguments

-atpg	Writes out the files needed to run Automatic Test Pattern Generation using the Encounter Test software.
-atpg_options {option1=value option2=value}	Specifies extra ATPG analysis options.
-build_model_options {option1=value option2=value}	Specifies extra options to apply when building the Encounter Test model.
-build_testmode_options {option1=value option2=value}	Specifies extra options to apply when building the test mode for Encounter Test.
<i>design</i>	Specifies the design for which to write out the Encounter Test input files. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
-directory <i>string</i>	Specifies the directory to which the output files must be written. <i>Default:</i> <i>current_working_directory/et_scripts</i>
-effort {low medium high}	Specifies the ATPG effort level to expend in resolving faults. Increasing effort will generally result in resolving more faults, but will require more processing time, sometimes significantly more. <i>Default:</i> low
-force	Specifies to continue even when the DFT logic appears to be incomplete.
-library <i>string</i>	Specifies the list of Verilog structural library files.

Command Reference for Encounter RTL Compiler

Design for Test

The Verilog libraries required to run Encounter Test ATPG and Incisive Enterprise simulation of the generated vectors must be provided using the `-library` option of the `write_et_rrfa` command. These libraries must be in a structural format. The files can be specified separately on the `write_et_rrfa` command line or can be referenced using an *include* file. Directories of Verilog files can also be specified but they cannot be referenced in the include file.

For example, if the Verilog files required to run ATPG and Incisive Enterprise simulation are:

```
./padcells.v  
./stdcells.v  
.memories/*.v  
.ip_blocks/*.v
```

`write_et_rrfa` can be used in either of the following ways:

1. If specifying files separately on the command line:

```
write_et_rrfa -library "./padcells.v ./stdcells.v \  
./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

And then specify the following:

```
write_et_rrfa -library "include_libraries.v \  
./memories ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where Encounter Test will be run.

`-rrfa_options string`

Specifies the extra options to run RRFA-based testability analysis in a string.

`-run_from_et_workdir`

Allows to run either Automatic Test Pattern Generator (ATPG) or Random Resistance Fault Analysis (RRFA) from the working directory specified with the `-directory` option.

Default: parent directory of the specified work directory.

Command Reference for Encounter RTL Compiler

Design for Test

`-verify_test_structures_options {option1=value option2=value}`

Specifies extra options to apply when performing test structure verification for Encounter Test.

Examples

- The following command generates the files to run an ATPG-based testability analysis.

```
write_et_rrfa -atpg -directory atpg -library $sim/mylib.v
```

Examining the `atpg` directory that was generated shows the following files:

```
rc:/> shell ls rrfra
run_compression_decomp_sim
run_compression_sim
runet.atpg
run_fullscan_sim
test.COMPRESSION_DECOMP.pinassign
test.COMPRESSION.pinassign
test.et_netlist.v
test.FULLSCAN.pinassign
test.rc_netlist.v
```

Related Information

See the following sections in *Design for Test in Encounter RTL Compiler*

- [Using Encounter Test to Automatically Select and Insert Test Points](#)
- [Exporting the Design for Test Point Selection](#)

write_io_speclist

```
write_io_speclist > iospeclist_file  
[-supplemental_file file]
```

Writes out the IOSpecList output file.

The IOSpecList output file describes the boundary scan architecture of the design. The file contains all ports (functional, test, and TAP) in the design, specifies the type and location of the boundary cells to be inserted on all the functional ports, and lists the instructions (both mandatory and user-defined) to be built in the JTAG Macro.

Options and Arguments

iospeclist_file Specifies the name of the file to be written.

-supplemental_file *file*

Specifies the name of the supplemental file to write out. This file and its corresponding IOSpecList file are used to define the boundary scan objects prior to inserting boundary scan logic. The supplemental file lists the boundary scan segments and the JTAG-instruction definitions for its related objects written to the IOSpecList file. When using an IOSpecList file to define the order of the boundary scan register, the supplemental file should be included into the RTL Compiler session before reading the IOSpecList input file. Both the supplemental and the IOSpecList files need only be written if your intention is to insert boundary-scan logic in a new RTL Compiler session.

Note: The recommended approach to completely restoring the DFT setup (including the boundary scan objects) in a new RTL Compiler session is to use the `write_script/read_netlist` approach.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Writing the IOSpecList in Design for Test in Encounter RTL Compiler](#)

Affected by these commands: [define_dft_jtag_instruction](#) on page 645

[define_dft_jtag_instruction_register](#) on page 649

[insert_dft_boundary_scan](#) on page 717

[insert_dft_mbist](#) on page 743

[insert_dft_ptam](#) on page 751

Related command: [read_io_speclist](#) on page 785

write_logic_bist_macro

```
write_logic_bist_macro -chains integer -channels integer  
    -max_length_of_channels integer  
    [-program_counter integer] [-capture_window_counter integer]  
    [-info_file string] [> file]
```

Writes out the structural netlist using generic logic for the LBIST macro. The generated macro has a PRPG, MISR, 1149.1 Interface, and a BIST Controller finite state machine. The macro can be initialized by the JTAG macro using the RUNBIST and SETBIST instructions.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features in Design for Test in Encounter RTL Compiler](#).

For more information about LBIST, contact your local Cadence representative.

Options and Arguments

-capture _window_counter integer

Specifies the value to be loaded into the capture window counter at the start of the runbist operation.

-chains integer

Specifies the number of top level scan data input and scan data output pairs. Typically, this is the number of uncompressed scan chains.

-channels integer

Specifies the number of compressed scan channels. It can include additional channels required to include the boundary scan chain in the macro too.

-info_file file

Specifies the file that contains more detailed information about the LBIST macro. It has details of the MISR and PRPG that can be used when writing out the scripts to run Encounter Test.

-max_length_of_channels integer

Specifies the length of the longest compressed scan channel.

-program_counter integer

Specifies the value to be loaded into the program counter at the start of the runbist operation.

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[Inserting LBIST Logic in Design for Test in Encounter RTL Compiler](#)

Related commands:

[insert_dft_logic_bist](#) on page 741

[write_et_lbist](#) on page 837

write_mbist_testbench

```
write_mbist_testbench
    [-create_embedded_test_options string]
    [-testbench_directory string]
    [-ncsim_library file_list]
    [-directory string] [design]
```

Writes out the necessary files and the template run scripts to create Verilog test benches to validate memory BIST, and executes the scripts using the Encounter Test software.

If your design contains ROMs, include the `rompath` and `romcontentsfile` keywords in the `create_embedded_test_options` string.

Typically a single memory BIST instruction set is implemented in the design. In this case, the `interfacefilelist` keyword to `create_embedded_test` is set by default from the `mbist_interface_files_location` design attribute. If multiple memory BIST instruction sets are implemented in the design, include the `interfacefilelist` keyword (indicating the interface files for a single instruction set) in the `create_embedded_test_options` string.

After inserting MBIST into the design and optionally a JTAG macro, write out the modified design to file. Then, use this command to generate an MBIST Verilog test bench for either a gate-level or RTL netlist to validate the MBIST functionality through simulation.

This command is designed to generate the following patterns by default:

- Bypass and production patterns—if JTAG control has been implemented for memory BIST in the design
- Poweron and burnin patterns—if direct access has been implemented for memory BIST in the design

These default generated test benches are created with a schedule intended to run all devices in parallel.

This command generates the following files prior to executing the run scripts in Encounter Test:

- `design_abstract.v`—A Verilog generic logic gates description of the netlist for Encounter Test
- `runet.write_mbist_testbench`—A template script file to run Create Embedded Test in Encounter Test, generating patterns for memory BIST
- `runet.write_vectors`—A template script file to run Write Vectors in Encounter Test to write the memory BIST patterns as Verilog test benches

Command Reference for Encounter RTL Compiler

Design for Test

- `irun.simscript.testbench_pattern_name`—One or more template scripts to compile and simulate the MBIST test bench pattern within the Incisive Enterprise simulator. By default, these scripts will be for BYPASS and Production patterns when JTAG is used, and for Poweron and Burnin patterns when direct access is used.

You can customize some files according to the setup requirements.

Note: To use this command you need an Encounter Test license. For more information on the exact product requirements, refer to [Encounter Test Product Requirements for Advanced Features](#) in *Design for Test in Encounter RTL Compiler*.

Options and Arguments

`-create_embedded_test_options {option1=value option2=value}`

Specifies extra options to apply when running Create Embedded Test in Encounter Test.

`design` Specifies the design for which to write out the Encounter Test input files.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

`-directory string` Specifies the directory to which the output files must be written.
Default: current_working_directory/wmt

`-ncsim_library file_list`

Specifies the list of library files required for the Incisive Enterprise simulation.

You can specify the files explicitly or you can specify an include file that lists the files. You can also specify directories of Verilog files but you cannot reference directories in an include file.

For example, if the Verilog files required to run simulation are:

```
./padcells.v  
./stdcells.v  
./memories/*.v  
./ip_blocks/*.v
```

`write_mbist_testbench` can be used in either of the following ways:

1. If specifying files separately on the command line:

Command Reference for Encounter RTL Compiler

Design for Test

```
write_mbist_testbench -ncsim_library "./padcells.v  
./stdcells.v ./memories ./ip_blocks" ...
```

2. If using an include file. Create an include file named `include_libraries.v` containing:

```
'include "./padcells.v"  
'include "./stdcells.v"
```

And then specify the following:

```
write_mbist_testbench -ncsim_library \  
"include_libraries.v ./memories ./ip_blocks" ...
```

Note: If you specify a relative path, the command interprets the path to be the location from where the Incisive Enterprise simulator will be run.

`-testbench_directory directory`

Specifies the directory to which the generated Verilog test benches must be written.

*Default: current_working_directory/
mbist_testbench*

Examples

- In the following example a single MBIST instruction set was implemented in the design:

```
write_mbist_testbench -testbench_directory ./mbist_verilog_testbenches \  
-ncsim_library ../$simulation_verilog_libraries \  
-directory ./mbist_verification_scripts
```

- The following example assumes you are working in a multiple block flow (MBIST was inserted on multiple designs or blocks) using separate JTAG instructions to access each block's MBIST engines.

```
write_mbist_testbench \  
-create_embedded_test_options \  
interfacefilelist="BLOCK_pattern_control.txt, \  
BLOCK_mbist_tdr_map.txt,BLOCK_mbistdiag_tdr_map.txt" \  
-testbench_directory ./mbist_verilog_testbenches \  
-ncsim_library../$simulation_verilog_libraries \  
-directory ./mbist_verification_scripts
```

- The following example shows how to specify the command when ROMS are present in the design:

```
write_mbist_testbench \  
-create_embedded_test_options \  
"rompath=./memory_data romcontentsfile=ROM256x34.hex" \  
-testbench_directory ./mbist_verilog_testbenches \  
-ncsim_library../$simulation_verilog_libraries \  
-directory ./mbist_verification_scripts
```

Command Reference for Encounter RTL Compiler

Design for Test

Related Information

[IDesign Flows](#) in “Inserting Memory Built-In-Self-Test Logic” in *Design for Test in Encounter RTL Compiler*

Affected by these commands

[define dft mbist direct access](#) on page 659
[insert dft boundary scan](#) on page 717
[insert dft jtag macro](#) on page 736
[insert dft mbist](#) on page 743

Related commands:

[write dft rtl model](#) on page 821
[write et bsv](#) on page 830
[write et mbist](#) on page 840
[write hdl](#) on page 255

Related attributes:

[dft rtl insertion](#)
[mbist instruction set](#)
[mbist interface files location](#)

write_scandef

```
write_scandef
  [-partition partition -chains chain [chain]...]
  [-version {5.4|5.5}]
  [-end_chains_before_lockups]
  [-dft_configuration_mode dft_configuration_mode_name]
  [-dont_split_by_library_domains]
  [-dont_split_by_power_domains]
  [-dont_use_timing_model_pins] [design] [> file]
```

Writes the scanDEF description of the top-level scan chains configured in the design for reordering using a physical design tool.

Options and Arguments

-chains <i>chain</i>	Specifies the scan chains that must be grouped in the same partition by the physical design tool. Use the <u>report_dft_chains</u> command to obtain a list of valid chain names. The tool ensures that chains or chain segments that are not compatible are not added to the same partition, but are further partitioned by adding the test clock name and test clock edge to the final partition name. Note: Requires version 5.5.
<i>design</i>	Specifies the design for which to write out the scanDEF description. If you omit the design name, the top-level design of the current directory of the design hierarchy is used.
-dft_configuration_mode <i>dft_configuration_mode_name</i>	Specifies the configuration mode for which to write the scan definition
-dont_split_by_library_domains	Indicates not to split the chains at the scan data input pin of the last flop in the originating (or from) library domain and at the scan data output pin of the first flop in the destination (or to) library domain. By default, the chains will be split based on the library domains. If the design has no library domains, the chains will not be split.

Command Reference for Encounter RTL Compiler

Design for Test

`-dont_split_by_power_domains`

Indicates not to split the chains at the scan data input pin of the last flop in the originating (or from) power domain and at the scan data output pin of the first flop in the destination (or to) power domain.

By default, the chains will be split based on the power domains. If the design has no power domains, the chains will not be split.

`-dont_use_timing_model_pins`

Prevents using the user-designated libcell timing model pins as the scanDEF chain START and STOP points. Instead an outward trace is performed to identify and use the first flip-flop scan data output pin and last flip-flop scan data input pin and use these pins as START and STOP pins in the scanDEF chains.

`-end_chains_before_lockups`

Terminates the scan segment at the scan data input pin of the scan flop which precedes the lockup element in the scan DEF chain.

`file`

Specifies the file to which the output must be written. To write out the scanDEF file in compressed format, specify a file name with the .gz extension.

Default: output is written to the screen

`-partition partition`

Specifies the name of a user-defined partition.

Note: Requires version 5.5.

`-version {5.4|5.5}`

Specifies which DEF version to write out. Version 5.5 writes out the MAXBITS and PARTITION keywords as regular statements (that is, uncommented).

Default: 5.4

Example

- The following example writes out the scanDEF information to the screen:

```
rc:/> write_scandef

VERSION 5.4 ;
NAMECASESENSITIVE ON ;
DIVIDERCHAR "/" ;
BUSBITCHARS "[]" ;
DESIGN top ;

SCANCHAINS 2 ;
- chain_1
+ START u_a/out_reg_1 Q
+ FLOATING
  u_a/out_reg_2 ( IN SI ) ( OUT Q )
  u_a/out_reg_3 ( IN SI ) ( OUT Q )
+ STOP buf_2 A
;

- chain_2
+ START buf_1 Y
+ FLOATING
  u_b/out_reg_0 ( IN SI ) ( OUT Q )
  u_b/out_reg_1 ( IN SI ) ( OUT Q )
+ STOP u_b/out_reg_3 SI
;

END SCANCHAINS
END DESIGN
```

Related Information

[Creating a scanDEF File in Design for Test in Encounter RTL Compiler](#)

Affected by this command: [connect scan chains](#) on page 620

Affected by these attributes: [Actual Scan Chain attributes](#)

Low Power Synthesis

- [build_rtl_power_models](#) on page 861
- [clock_gating](#) on page 863
- [clock_gating_connect_test](#) on page 865
- [clock_gating_declone](#) on page 866
- [clock_gating_import](#) on page 867
- [clock_gating_insert_in_netlist](#) on page 869
- [clock_gating_insert_obs](#) on page 870
- [clock_gating_join](#) on page 872
- [clock_gating_remove](#) on page 874
- [clock_gating_share](#) on page 876
- [clock_gating_split](#) on page 878
- [read_saif](#) on page 880
- [read_tcf](#) on page 885
- [read_vcd](#) on page 890
- [report_clock_gating](#) on page 893
- [report_operand_isolation](#) on page 894
- [report_power](#) on page 895
- [state_retention](#) on page 896
- [state_retention_connect_power_gating_pins](#) on page 897
- [state_retention_swap](#) on page 898
- [write_forward_saif](#) on page 899

Command Reference for Encounter RTL Compiler

Low Power Synthesis

- [write_saif](#) on page 901
- [write_tcf](#) on page 903

build_rtl_power_models

```
build_rtl_power_models
  [-clean_up_netlist]
  [-clock_gating_logic]
  [-relative instance_list]
  [-design design]
```

Builds detailed power models for more accurate RTL power analysis. The models are used in subsequent RTL power analysis reports.

The power models are MSV and PSO aware.

If you have super-threading enabled, it will be used for power model building.

Options and Arguments

<code>-clean_up_netlist</code>	Requests to remove unreachable logic in the netlist as this can affect the accuracy of the estimation. Note: Unreachable logic is removed from the input netlist, without changing the logic functionality.
<code>-clock_gating_logic</code>	Requests to include power estimation for clock-gating logic. Note: Power estimation of clock-gating components can increase runtime considerably.
<code>-design design</code>	Specifies the design for which to build the power models.
<code>-relative instance_list</code>	Builds separate power models for each of the specified hierarchical instances. Models for the top design are built separately at the end.

Example

The following example shows an extract of the messages that are printed in the log file when you build the RTL power models.

```
rc:/> build_rtl_power_models -clean_up_netlist -clock_gating_logic
Cleaning up the design /designs/mult_bit_muxed_add ...
Starting building RTL power analysis models ...
Preprocessing the netlist for building RTL power models ...
Building RTL power models for top-level design /designs/mult_bit_muxed_add ...
Building power models for clock gating logic ..
Done building models for power analysis.
RTL power modeling has finished. Use command 'report power' to see power report.
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

The following example shows the messages that are printed in the log file when you build separate RTL power models for hierarchical instances.

```
rc:/> build_rtl_power_models -clean_up_netlist -relative {mult_1 mult_2}
Cleaning up the design /designs/test ...
Starting building RTL power analysis models ...
Preprocessing the netlist for building RTL power models ...
Building RTL power models for domain /designs/test/instances_hier/mult_1 ...
Building RTL power models for domain /designs/test/instances_hier/mult_2 ...
Building RTL power models for top-level design /designs/test ...
Done building models for power analysis.
RTL power modeling has finished. Use command 'report power' to see power report.
```

Related Information

[RTL Power Analysis in Low Power in Encounter RTL Compiler](#)

Related command: [report power](#) on page 895

Affected by this attribute: [lp_insert_clock_gating](#)

clock_gating

```
clock_gating
{ connect_test | declone | import | insert_in_netlist
| insert_obs | join | remove | share | split}
```

Manipulates a netlist for clock gating.



The `clock_gating` commands only work on a mapped netlist.

Options and Arguments

<code>connect_test</code>	Connects the test input of all clock-gating logic.
<code>declone</code>	Merges clock-gating instances driven by the same inputs.
<code>import</code>	Processes clock-gating instances that were either manually inserted or inserted by third-party tools to make them recognizable as clock-gating instances by the RC-LP engine.
<code>insert_in_netlist</code>	Inserts clock-gating logic on a mapped netlist.
<code>insert_obs</code>	Inserts and connects observability logic.
<code>join</code>	Joins multiple-stage clock-gating logic into a single clock-gating instance with a complex enable function.
<code>remove</code>	Removes the specified clock-gating logic.
<code>share</code>	Extracts the enable function shared by clock-gating logic and inserts shared clock-gating logic with the common enable sub function as the enable signal.
<code>split</code>	Splits a single clock-gating instance with a complex enable function into multiple stages of clock-gating logic.

Related Information

Related commands:	<u>clock_gating connect test</u> on page 865 <u>clock_gating declone</u> on page 866 <u>clock_gating import</u> on page 867
-------------------	---

Command Reference for Encounter RTL Compiler

Low Power Synthesis

[clock gating insert in netlist](#) on page 869

[clock gating insert obs](#) on page 870

[clock gating join](#) on page 872

[clock gating remove](#) on page 874

[clock gating share](#) on page 876

[clock gating split](#) on page 878

clock_gating connect_test

```
clock_gating connect_test
```

Globally connects the test input of all clock-gating logic to the test signal specified through the `lp_clock_gating_test_signal` attribute and marks this network as ideal. This command applies to the current design or the current hierarchical instance.

If the clock-gating test input is already connected, the command has no effect.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Example

- The following command connects the test inputs connected to Test1.

```
synthesize
...
set_att lp_clock_gating_test_signal Test1
...
clock_gating connect_test
```

Related Information

[Clock Gating with DFT in Low Power in Encounter RTL Compiler](#)

[Scan Insertion after Clock-Gating Insertion in Low Power in Encounter RTL Compiler](#)

Related command: [report clock_gating](#) on page 893

Affected by the attribute: [lp_clock_gating_test_signal](#)

clock_gating declone

```
clock_gating declone  
    [-hierarchical] [-no_clock_tree_traversal]
```

Merges clock-gating instances driven by the same inputs. The RC-LP engine automatically removes any dangling ports.

Note: This command works only on a netlist whose clock-gating logic was inserted by the RC-LP engine.

Options and Arguments

-hierarchical Allows traversing the design hierarchy to search for clock-gating instances that can be merged. As a result, new ports can be added for the gated-clock signal.

By default, this command only affects instances at the current level of the hierarchy.

-no_clock_tree_traversal

Prevents traversing through buffers and inverter pairs on the clock signal.

If you do not set this option, RTL Compiler can remove buffers or inverter pairs on the clock path during optimization unless the buffers or inverter pairs are marked preserved.

Related Information

[Decloning Clock-Gating Instances in Low Power in Encounter RTL Compiler](#)

Related command: [report clock_gating](#) on page 893

Affected by this attribute: [lp_clock_gating_max_flops](#)

clock_gating import

```
clock_gating import  
[-start_from instance] [-hierarchical] [-detail]
```

Processes clock-gating instances that were either manually inserted or inserted by third-party tools to make them recognizable as clock-gating instances by the RC-LP engine.

The command returns the total number of instances imported.

Currently, the RC-LP engine recognizes the following structures as clock-gating instance:

- Two-input AND or NAND gates that have a clock signal driving one of the inputs
 - In this case, the other pin is assumed to be the enable pin.
Note: If the other pin is part of the test network, the RC-LP engine does not recognize the gate as a clock-gating instance.
- Integrated clock-gating cells

Currently, the RC-LP engine does not recognize these structures as clock-gating instances if they were defined in a separate module that is instantiated in the netlist.

The RC-LP engine creates a new hierarchical instance (RC(CG)_HIER_INST) for each clock-gating instance it recognizes and adds it to the list of clock-gating instances that the RC-LP engine has created.

Options and Arguments

-detail Prints a summary with the names of the imported clock-gating modules and the number of instances found for each clock-gating module.

-hierarchical Allows traversing the design hierarchy to process clock-gating instances that were not inserted by RTL Compiler.

By default, this command only affects instances at the current level of the design hierarchy.

-start_from *instance*

Starts processing clock-gating instances that were not inserted by RTL Compiler from the specified hierarchical instance.

By default, the process starts from the current location in the design hierarchy.

Examples

- The following example shows the minimum information listed when clock-gating instances are successfully imported. The number “2” is the total number of instances imported.

```
rc:/> clock_gating import -start_from /designs/top -hier  
Importing clock_gating logic from /designs/top  
Imported 2 Clock Gating instances
```

2

- The following example shows the additional information listed when the **-detail** option is specified.

```
rc:/> clock_gating import -start_from /designs/top -hier -detail  
Importing clock_gating logic from /designs/top
```

Detailed report for clock_gating import	
Module name	Import count
a_1	1
a	1

```
Imported 2 Clock Gating instances
```

2

Related Information

Related command:

[report clock_gating](#) on page 893

clock_gating insert_in_netlist

`clock_gating insert_in_netlist`

Inserts clock-gating logic in a mapped netlist if the D-input of a flip-flop is driven by a two-input MUX and there is a feedback loop from the Q-output to the D-input through one of the data pins of the two-input MUX.

You should only use this command on a netlist that was already mapped (possibly by a third-party tool).

If you set the `lp_clock_gating_test_signal` attribute before you enter this command, the RC-LP engine can connect the test-control signal to the test pins of the clock-gating logic during clock-gating insertion.

Note: This command allows the flip-flops and MUX logic to be in different hierarchies.

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Clock Gating and Scan Chain Insertion in Mapped Netlist](#)
- [Clock-Gating Insertion in a Scan-Connected Netlist](#)

Related command: [report clock_gating](#) on page 893

Affected by this attribute: [lp_clock_gating_test_signal](#)

clock_gating insert_obs

```
clock_gating insert_obs
    [-hierarchical] [-make_obs_module]
    [-max_cg integer]
    [-ignore_clock_constraint]
    [-exclude instance...]
    [-disable_clock -libcell libcell]
```

Inserts and connects circuitry to improve the observability of the design after clock-gating logic is inserted. This command applies to the current design or the current hierarchical instance.

Note: To make sure that the enable signal of the clock-gating logic is observable, set the `lp_clock_gating_add_obs_port` design attribute to `true` before you insert the clock-gating logic.

Observability logic is inserted based on clock information. The clock information is required because only clock-gating logic driven by the same clock can share an observation flip-flop. The clock information can be derived from clock constraints or from the physical connectivity.

Note: This command works on a netlist whose clock-gating logic was either inserted or imported by the RC-LP engine.

Options and Arguments

<code>-disable_clock</code>	Specifies to gate the clock of the observability flip-flops. Note: One gating cell is inserted per flip-flop. The RC-LP engine creates a separate subdesign for each gating cell.
<code>-exclude instance</code>	Prevents insertion of observability logic in the specified hierarchical instances.
<code>-hierarchical</code>	Allows insertion and connection of observability logic in the current level of the hierarchy and all its children. By default, this command only affects the current level of the hierarchy.
<code>-ignore_clock_constraint</code>	Inserts observability logic based on physical connectivity. By default, observability logic is inserted based on clock constraints defined with the <code>define_clock</code> command.
<code>-libcell libcell</code>	Specifies the name of a library cell to be used for gating.

Command Reference for Encounter RTL Compiler

Low Power Synthesis

Note: You can only specify an AND cell to gate the observability logic.

`-make_obs_module` Creates a separate hierarchy (module) for each observation flip-flop and its associated XOR tree.

`-max_cg integer` Specifies the maximum number of clock-gating cells that can be observed per observation flip-flop. Specify an integer between 1 and 32.

Default: 8

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Clock Gating with DFT](#)
- [Clock Gating and Scan Chain Insertion in Mapped Netlist](#)
- [Recommended Bottom-Up Clock Gating Flow with DFT](#)
- [Scan Insertion after Clock-Gating Insertion](#)
- [Clock-Gating Insertion in a Scan-Connected Netlist](#)

Related commands: [define_clock](#) on page 290

[report_clock_gating](#) on page 893

Affected by this attribute: [lp_clock_gating_add_obs_port](#)

clock_gating join

```
clock_gating join  
  [-hierarchical] [-max_level integer]  
  [-multi_fanouts] [-start_from instance]
```

Combines multiple stages of clock-gating logic into a single clock gating instance with a complex enable function.

Note: This command works on a netlist whose clock-gating logic was either inserted or imported by the RC-LP engine.

Options and Arguments

<code>-hierarchical</code>	Allows joining of clock-gating logic down the hierarchy starting from the current directory. By default, this command only affects the current level of the hierarchy.
<code>-max_level integer</code>	Specifies the maximum levels of clock-gating instances that can be combined. If you specify n, n+1 stages can be combined. <i>Default:</i> 1 allowing 2 levels to be combined.
<code>-multi_fanouts</code>	Allows joining even if the root stage clock-gating instance is driving multiple clock-gating instances.
<code>-start_from instance</code>	Joins the clock-gating logic starting from the specified hierarchical instance.

Examples

- The following command allows joining clock-gating instances across the hierarchy of hierarchical instance i1.

`clock_gating join -hierarchical -start_from [find / -inst i1]`
- The following command allows joining three stages of clock-gating instances across the hierarchy starting from the current directory.

`clock_gating join -hierarchical -max_level 2`

Command Reference for Encounter RTL Compiler

Low Power Synthesis

Related Information

[Consolidating Multi-Stage Clock-Gating Logic in Low Power in Encounter RTL Compiler](#)

Related command: [report clock_gating](#) on page 893

clock_gating remove

```
clock_gating remove
  [ -hierarchical [-obs_only]
  | -cg_list instance_list [-obs_only]
  | -flops flops
  [ -no_verbose]
  [ -effort {low|high}]
```

Removes clock-gating logic inserted by the RC-LP engine from the current design or the current hierarchical instance.

Note: This command works on a netlist whose clock-gating logic was either inserted or imported by the RC-LP engine.

Options and Arguments

-cg_list *instance_list*

Specifies a list of clock-gating instances to be removed. Use a full path name to identify these instances.

Note: If you specify a clock-gating instance with an incomplete path, the tool searches for that instance from the root of the design hierarchy and might select multiple instances with the same name from different hierarchies.

-effort {high|low} Specifies the effort level.

Choosing low effort results in better runtime performance but at the cost of an area increase. In this case, the RC-LP engine reconstructs the original MUX and feedback loop from the flip-flop to the MUX.

For large designs high effort can result in long runtimes, but the feedback logic is optimized.

Default: low

-flops *flops* Removes clock gating from the specified flops (that is, recreates the feedback loop for those flops).

If you specified all flops that are gated by the same clock-gating instance, the clock-gating instance will also be removed.

-hierarchical Removes all clock-gating logic in the hierarchy of the current design or subdesign.

By default, this command only affects the current level of the hierarchy.

- | | |
|-------------|---------------------------------------|
| -no_verbose | Suppresses info and warning messages. |
| -obs_only | Removes only the observability logic. |

Example

- The following command removes all clock-gating instances in the hierarchy of subdesign sub1.

```
rc:/designs/alu/subdesigns/sub1> clock_gating remove -hier
```

- The following command removes the clock-gating instances RC(CG)_HIER_INST_121 and RC(CG)_HIER_INST_122 from the current design hierarchy.

```
rc:/> clock_gating remove -cg_list \
/designs/top/instances_hier/RC(CG)_HIER_INST_121 \
/designs/top/instances_hier/RC(CG)_HIER_INST_122
```

```
Clock-gating instance removed /designs/top/instances_hier/RC(CG)_HIER_INST_121
Clock-gating instance removed /designs/top/instances_hier/RC(CG)_HIER_INST_122
```

Related Information

[Removing Clock-Gating Instances in Low Power in Encounter RTL Compiler](#)

Related command: [report clock_gating](#) on page 893

clock_gating share

```
clock_gating share  
  [-hierarchical] [-max_level integer]  
  [-max_stage {integer|string}]
```

Extracts the enable function shared by clock-gating logic and inserts shared clock-gating logic with the common enable sub function as the enable signal. The resulting netlist has multiple stages of clock-gating logic.

Note: This command works on a netlist whose clock-gating logic was either inserted or imported by the RC-LP engine.

Options and Arguments

-hierarchical Inserts shared clock-gating logic down the hierarchy starting from the design or current hierarchical instance.

By default, this command only affects the current level of the hierarchy.

-max_level integer Specifies the maximum levels of logic (buffers and inverters excluded) to traverse in the enable fanin of clock-gating instances to extract the common enable function.

Default: 5

-max_stage {integer|string}

Specifies the maximum number of stages of shared clock-gating logic.

To specify the same maximum number of stages for all clocks, specify an integer.

To specify the maximum number of stages per clock, use a string. The string must have the following format:

{*clock integer*} {*clock integer*} ...}

If this option is not specified, no limit is applied to the number of stages.

Examples

- The following command allows sharing clock-gating logic across the hierarchy starting from the current directory and allows traversing two levels of logic to extract the common enable function.

```
clock_gating share -hierarchical -max_level 2
```

- The following command will insert a maximum of 2 stages of shared clock-gating logic for clock clk1 and a maximum of 3 stages of clock-gating logic for clock clk2.

```
clock_gating share -max_stage { {clk1 2} {clk2 3} }
```

Related Information

[Creating Shared Clock Gating Logic Using Common Enable in Low Power in Encounter RTL Compiler](#)

Related command:

[report clock_gating](#) on page 893

clock_gating split

```
clock_gating split  
  [-hierarchical] [-max_level integer]  
  [-power_driven] [-start_from instance]
```

Splits a single clock gating instance with a complex enable function into multiple stages of clock-gating logic.

Note: This command works on a netlist whose clock-gating logic was either inserted or imported by the RC-LP engine.

Options and Arguments

-hierarchical Allows splitting of clock-gating logic down the hierarchy starting from the current directory.

By default, this command only affects the current level of the hierarchy.

-max_level *integer* Specifies how many times a complex enable function can be split.

Default: 1 allowing the complex enable function to be split into two stages.

-power_driven Forces to use the signal with smallest toggle rate as the root-level enable.

By default, the RC-LP engine considers timing first and uses the late signal as the root-level enable.

-start_from *instance*

Splits the clock-gating logic starting from the specified hierarchical instance.

Examples

- The following command allows splitting clock-gating instances across the hierarchy of hierarchical instance i1.

```
clock_gating split -hierarchical -start_from [find / -inst i1]
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

- The following command allows splitting a single clock-gating instance with a complex enable function into three stages of clock-gating instances across the hierarchy starting from the current directory.

```
clock_gating split -hierarchical -max_level 2
```

Related Information

[Splitting Clock-Gating Instances into Multiple Levels of Clock-Gating Logic in Low Power in Encounter RTL Compiler](#)

Related command: [report clock_gating](#) on page 893

Command Reference for Encounter RTL Compiler

Low Power Synthesis

read_saif

```
read_saif [-scale scale_factor]  
          [-update [-weight weight_factor]]  
          [-verbose] [-instance instance] file
```

Reads switching activity information in Synopsys switching activity interchange format (SAIF) and converts it internally to the Toggle Count Format (TCF) for power estimation.

The `read_saif` command can read files that have been compressed with gzip (`.gz` extension). The `.gz` file is unzipped in memory while the file is read in.

Note: If you read in subsequent SAIF files without the `-update` option, only the probability values and toggle counts of the pins and nets in the current SAIF file are overwritten. The other net values remain unchanged.

The following applies when *updating* the probability values and toggle counts:

- If the probability values and toggle rates were not previously user asserted, the updated probability and toggle rates are determined by the values specified in the SAIF file.
 - If the probability and toggle rates were previously user asserted, the new probability and toggle rates are calculated as follows:

```

prob_new = (prob_old + w * prob_spec) / (1+w)
tr_new = (tr_old + w * tc_spec / duration_spec) / (1+w)

```

where `prob_old` and `tr_old` are the stored values, and `prob_spec`, `tc_spec`, and `duration_spec` are the probably, toggle count, and duration values derived from the new SAIF file.

Options and Arguments

file Specifies the name of the SAIF file. The file can have any name, suffix, or length.

-instance *instance* Reads in the switching activities for the specified instance.

The instance name can refer to an instance in the design loaded in RC, or can refer to an instance name in the SAIF file.

- If the instance name refers to an instance in the design loaded in RTL Compiler, the RC-LP engine asserts switching activities on that instance in the loaded design.

Command Reference for Encounter RTL Compiler

Low Power Synthesis

In this case, a *complete* design is loaded in RTL Compiler. However, the SAIF file is incomplete and contains only switching activities for the specified instance.

- If the instance name refers to an instance in the SAIF file, the RC-LP engine asserts switching activities on the design loaded in RTL Compiler.

The design loaded in RTL Compiler is

- a *partial* design if the top-level instance in the SAIF file corresponds to the full design, while the *specified* instance is a lower-level instance.
- the full design if the *specified* instance in the SAIF file corresponds the top-level design scope in the SAIF file.

In this case, the SAIF file is a hierarchical SAIF and contains switching activities for the full design.

Note: The name of the instance in the SAIF file does not need to match the name of the design.

-scale *scale_factor*

Scales the toggle counts in the SAIF file by dividing them by the specified factor. Use a positive (non-zero) floating number.

Default: 1.0

-update

Indicates that you are updating the probability values and toggle counts.

-verbose

Prints a message for each net that is asserted.

Default: Silent mode. Prints the percent completion messages.

-weight *weight_factor*

Specifies the relative weight of the probability values and toggle rates in the new SAIF file with respect to the probability values and toggle rates currently stored in the design. Use a positive floating number. This option is only valid with the -update option.

Default: 1.0

Command Reference for Encounter RTL Compiler

Low Power Synthesis

Examples

For the following examples, consider the following SAIF file (`and2.saif`):

```
(SAIFFILE
(SAIFVERSION "2.0")
(DIRECTION "backward")
(DESIGN "a")
(DATE "date")
(VENDOR "Cadence Design Systems Inc.")
(PROGRAM NAME "program")
(VERSION "version")
(DIVIDER / )
(TIMESCALE 1 ns)
(DURATION 1000.00)
(INSTANCE a
  (NET
    (in2
      (T0 700) (T1 300) (TC 16)
    )
    ("in1"
      (T0 900) (T1 100) (TC 9)
    )
    (out
      (T0 100) (T1 900) (TC 7)
    )
  )
)
)
```

- The following command reads the SAIF file with the `-verbose` option:

```
rc:/> read_saif and2.saif -verbose
10.0 % done
...
60.0 % done
  Setting attribute of net 'in2': 'lp_asserted_probability' = 0.30000
  Setting attribute of net 'in2': 'lp_asserted_toggle_rate' = 0.016000
70.0 % done
  Setting attribute of net 'in1': 'lp_asserted_probability' = 0.10000
  Setting attribute of net 'in1': 'lp_asserted_toggle_rate' = 0.009000
80.0 % done
  Setting attribute of net 'out': 'lp_asserted_probability' = 0.90000
  Setting attribute of net 'out': 'lp_asserted_toggle_rate' = 0.007000
90.0 % done
Nets/ports asserted in SAIF file : 3
Total Nets/ports in SAIF file   : 3
-----
Asserted Primary inputs in design          : 2 (100.00%)
Total connected primary inputs in design    : 2 (100.00%)
-----
Asserted sequential outputs                 : 0 (0%)
Total connected sequential outputs         : 0 (100.00%)
-----
Total nets in design                      : 4 (100.00%)
Nets asserted                           : 3 (75.00%)
Clock nets                               : 0 (0.00%)
Constant nets                            : 0 (0.00%)
Nets with no assertions                  : 1 (25.00%)
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

- The following command scales the toggle counts in the SAIF file by a factor 2:

```
rc:/> read_saif and2.saif -scale 2.0
```

Check the asserted toggle rates on nets in1 and in2:

```
rc:/> get_attr lp_asserted_toggle_rate nets/in1  
0.004500  
rc:/> get_attr lp_asserted_toggle_rate nets/in2  
0.008000
```

- In the following example, assume you have read in the `and2.saif` file, and you read in the following `and2_new.saif` file:

```
(SAIFILE  
...  
(TIMESCALE 1 ns)  
(DURATION 1000.00)  
(INSTANCE a  
  (NET  
    (in1  
      (T0 900) (T1 100) (TC 5)  
    )  
  )  
)
```

The following command updates the stored switching activities with the data in the `and2_new.saif` and gives a two times higher weight on the values in the `and2_new.saif` file.

```
read_saif -update -weight 2 and2_new.saif
```

Check the asserted toggle rates on nets in1:

```
rc:/> get_attr lp_asserted_toggle_rate nets/in1  
0.006333
```

This can be calculated as follows:

$$\begin{aligned} tr_{\text{new}} &= (tr_{\text{old}} + w * tc_{\text{spec}} / duration_{\text{new}}) / (1+w) \\ &= (0.009000 + 2 * 0.005000) / (1+2) = 0.006333 \end{aligned}$$

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Reading Switching Activity Information from a SAIF File](#)
- [Checking System Messages when Reading Switching Activities](#)

Affects this command: [report power](#) on page 895

Related command: [write_saif](#) on page 901

Sets these attributes: [lp_asserted_probability](#)

Command Reference for Encounter RTL Compiler

Low Power Synthesis

[lp_asserted_toggle_rate](#)

Related attributes:

[lp_probability_type](#)

[lp_toggle_rate_type](#)

Command Reference for Encounter RTL Compiler

Low Power Synthesis

read_tcf

```
read_tcf [-scale scale_factor]  
        [-update [-weight weight_factor]]  
        [-verbose] [-instance instance]  
        [-ignorecase] file
```

Reads or updates probability values and toggle counts of the pins and nets in the specified Toggle Count Format (TCF) file and stores the assertions as pin or net attributes, so they can be used for power estimation and optimization.

The `read_tcf` command can read files that have been compressed with gzip (.gz extension). The .gz file is unzipped in memory while the file is read in.

Note: If you read in subsequent TCF files without the `-update` option, only the probability values and toggle counts of the pins and nets in the current TCF file are overwritten. The other net values remain unchanged.

When *updating* the probability values and toggle counts, the new probability and toggle rates are calculated as follows:

```

prob_new = (prob_old + w * prob_spec) / (1+w)
tr_new = (tr_old + w * tc_spec/duration_spec) / (1+w)

```

where

- `prob_old` and `tr_old` are either the user-asserted values or the values computed using the power simulation engine.
 - `prob_spec`, `tc_spec`, and `duration_spec` are the probability, toggle count, and duration values specified in the new TCF file.

Options and Arguments

file Specifies the name of the TCF file. The file can have any name, suffix, or length.

-ignorecase Ignores the case of module, net, and pin names in the TCF file when searching for the matching module, net, or pin in the design.

By default, case is taken into account.

Note: Using this option might result in increased run time.

-instance *instance*

Reads in the switching activities for the specified instance.

Command Reference for Encounter RTL Compiler

Low Power Synthesis

The instance name can refer to an instance in the design loaded in RC, or can refer to an instance name in the TCF file.

- If the instance name refers to an instance in RTL Compiler, the RC-LP engine asserts switching activities on that instance in the loaded design.

In this case, a *complete* design is loaded in RTL Compiler. However, the TCF file is incomplete and contains only switching activities for the specified instance.

- If the instance name refers to an instance in the TCF file, the RC-LP engine asserts switching activities on the design loaded in RTL Compiler.

The design loaded in RTL Compiler is

- a *partial* design if the top-level instance in the TCF file corresponds to the full design, while the *specified* instance is a lower-level instance.
- the full design if the *specified* instance in the TCF file corresponds to the top-level design scope in the TCF file.

In this case, the TCF file is a hierarchical TCF and contains switching activities for the full design.

Note: The name of the instance in the TCF file does not need to match the name of the design.

`-scale scale_factor`

Scales the toggle counts in the TCF file by dividing them by the specified factor. Use a positive (non-zero) floating number.

Default: 1.0

`-update`

Indicates to update the probability values and toggle counts.

`-verbose`

Prints a message for each net that is asserted.

Default: Silent mode. Prints the percent completion messages.

`-weight weight_factor`

Specifies the relative weight of the probability values and toggle rates in the new TCF file with respect to the probability values and toggle rates currently stored in the design. Use a positive floating number. This option is only valid with the -update.

Default: 1.0

Command Reference for Encounter RTL Compiler

Low Power Synthesis

Examples

- Consider the following TCF file (example1.tcf):

```
tcffile () {
    tcfversion : "1.0";
    duration : "1.500000e+05";
    unit : "ns";
    instance () {
        pin () {
            "i_12/Z" : "0.566 747";
            "n_n1/B" : "0.516 475";
            "hier1/i_0/Z" : "0.5 500";
            "hier1/n_n0/Z" : "0.5 500";
            "hier1/n_n0/A" : "0.5 500";
            "hier1/i_0/A" : "0.5 500";
            "hier1/i_0/B" : "0.5 500";
            "n_n1/A" : "0.61 516";
        }
    }
}
```

To read this TCF file (example1.tcf), use the following command:

```
rc:/> read_tcf example1.tcf
```

- In the following TCF file (example2.tcf), the only difference with the previous TCF file is that the duration in example2.tcf is half of the duration in example1.tcf.

```
tcffile () {
    tcfversion : "1.0";
    duration : "0.75000e+05";
    unit : "ns";
    instance () {
        pin () {
            "i_12/Z" : "0.566 747";
            "n_n1/B" : "0.516 475";
            "hier1/i_0/Z" : "0.5 500";
            "hier1/n_n0/Z" : "0.5 500";
            "hier1/n_n0/A" : "0.5 500";
            "hier1/i_0/A" : "0.5 500";
            "hier1/i_0/B" : "0.5 500";
            "n_n1/A" : "0.61 516";
        }
    }
}
```

To make the toggle rates on all pins the same as in the previous example, use the following command:

```
rc:/> read_tcf -scale 2.0 example2.tcf
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

- Consider the following TCF file (example1.tcf):

```
tcffile () {
    tcfversion : "1.0";
    duration   : "1.000000e+05";
    unit       : "ns";
    instance () {
        pin () {
            "i_0/A"      : "0.5 500";
            "i_0/B"      : "0.6 600";
            "i_0/Z"      : "0.7 700";
        }
    }
}
```

Assume you read this TCF file with the following command:

```
rc:> read_tcf example1.tcf
```

Now consider the following TCF file (example2.tcf):

```
tcffile () {
    tcfversion : "1.0";
    duration   : "1.500000e+05";
    unit       : "ns";
    instance () {
        pin () {
            "i_0/A"      : "0.5 600";
            "i_0/B"      : "0.6 750";
            "i_0/Z"      : "0.7 900";
        }
    }
}
```

Assume you read this TCF file with the following command:

```
rc:> read_tcf -update -weight 0.5 example2.tcf
```

You would get the same result by:

- a. Creating the following TCF file (example3.tcf)

```
tcffile () {
    tcfversion : "1.0";
    duration   : "1.500000e+05";
    unit       : "ns";
    instance () {
        pin () {
            "i_0/A"      : "0.5 700";
            "i_0/B"      : "0.6 850";
            "i_0/Z"      : "0.7 1000";
        }
    }
}
```

- b. Reading the example3.tcf file using the following command:

```
read_tcf example3.tcf
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Reading Switching Activity Information from a TCF File](#)
- [Checking System Messages when Reading Switching Activities](#)

[TCF Syntax in Toggle Count Format Reference.](#)

Affects this command: [report power](#) on page 895

Sets these attributes: [lp_asserted_probability](#)

[lp_asserted_toggle_rate](#)

Related attributes: [lp_probability_type](#)

[lp_toggle_rate_type](#)

read_vcd

```
read_vcd
  [-static
   | -activity_profile [-time_window time]
   [-simvision] [-write_sst2 file] ]
  [-start_time start_monitoring_time]
  [-end_time end_monitoring_time]
  [-module {design|subdesign}] [-vcd_module module]
  [-ignorecase] vcd_file
```

Reads a Value Change Dump (VCD) file for power analysis. You can

- Perform static power analysis
- Build an activity profile

Note: If no options are specified, static power analysis is performed by default.

The `read_vcd` command can read files that have been compressed with gzip (.gz extension). The .gz file is unzipped while the file is read in.

Options and Arguments

`-activity_profile` Builds a profile of the activities for the set scope without annotating the switching activities to the design.

By default, profiling is done for the whole design. You can limit the scope to a portion of the design by setting the `lp_dynamic_analysis_scope` attribute to `true` on those instances for which you want to build the profile.

The RC-LP engine captures the toggle count of objects within a given time window. The object can be a net, pin or a hierarchical instance. For a hierarchical instance, the activity will be the sum of the activities of the objects in that instance.

Note: By default, the `read_vcd` command performs static power analysis if neither the `-static` or `-activity_profile` option was specified.

`-end_time end_monitoring_time`

Specifies the time you want to end monitoring the switching activities or events. Specify a value larger than zero in picoseconds.

By default, the activities or events are monitored till the end (last timestamp of the VCD file).

Command Reference for Encounter RTL Compiler

Low Power Synthesis

`-ignorecase` Ignores the case of module, net, and pin names in the VCD file when searching for the matching module, net, or pin in the design.

By default, case is taken into account.

Note: Using this option might result in increased run time.

`-module {design | subdesign}`

Specifies the name of the design or subdesign in the RTL Compiler hierarchy to which the parsed VCD hierarchy (specified through the `-vcf_module` option) corresponds.

For example, if a *partial* design is loaded in RTL Compiler but you have a VCD file that contains switching activities for the full design, the top design in RTL Compiler will correspond to an instance in the VCD hierarchy.

You can also have a full design loaded in RTL Compiler, but only have a partial VCD. In that case you need to specify the name of the subdesign in the RTL Compiler hierarchy to which the VCD file applies.

By default, the VCD file applies to the top design in the RTL Compiler hierarchy. If multiple top designs exists, you must specify the name of the top design.

`-simvision`

Invokes SimVision to display the activity profile.

Note: To use this option you need to have SimVision installed and your operating system PATH environment variable must include the path to SimVision.

`-start_time start_monitoring_time`

Specifies the time you want to start monitoring the switching activities or events. Specify a value larger than zero in picoseconds.

By default, the first timestamp in the VCD file is considered as the start time to monitor.

`-static`

Calculates the switching activities of each of the nets and pins from the time you want to start monitoring the switching activities to the time you want to stop monitoring, and then stores the information as assertions on the nets and pins.

By default, the `read_vcd` command performs static power analysis if neither the `-static` or `-activity_profile` option was specified.

Command Reference for Encounter RTL Compiler

Low Power Synthesis

`-time_window window`

Specifies the time increment, in picoseconds, in which you want the RC-LP engine to divide the period during which the events are monitored. The specified time window must be larger than zero.

By default, the time window is calculated based on the setting of the `lp_power_analysis_effort` root attribute and the values of the `-start_time` and `-end_time` options.

`vcd_file`

Specifies the name of the value change dump (VCD) file.

`-vcd_module module`

Starts parsing the VCD hierarchy from the specified module. You can specify to start parsing from the top or from a particular module in the VCD hierarchy.

Default: first scope encountered is used as the top scope

`-write_sst2 file`

Specifies the prefix of the SST2 database files to generate to view the data in other waveform viewers.

Example

- The following command reads `my_vcd.vcd`, generates an activity profile from 10 to 100 ps based on a time window of 10ps, and invokes SimVision to display the activity profile.

```
read_vcd -vcd_module mid2 -activity_profile -start_time 10 -end_time 100 \
-time_window 10 -simvision my_vcd.vcd
```

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Reading Switching Activity Information from a VCD File](#)
- [Checking System Messages when Reading Switching Activities](#)

Affects this command:

[report power](#) on page 895

Related attributes:

[lp_dynamic_analysis_scope](#)

[lp_power_analysis_effort](#)

report clock_gating

Refer to [report clock_gating](#) in [Chapter 8, “Analysis and Report.”](#)

report operand_isolation

Refer to [report operand_isolation](#) in [Chapter 8, “Analysis and Report.”](#)

report power

Refer to [report power](#) in [Chapter 8, “Analysis and Report.”](#)

state_retention

```
state_retention
  {connect_power_gating_pins | swap}
```

Defines the aspects of mapping to state retention cells.

Options and Arguments

connect_power_gating_pins

Connects the power gating pins in a mapped netlist.

swap

Replaces sequential cells with their equivalent state retention power gating cells in a mapped netlist.

Related Information

[State-Retention Cell Replacement when Starting with Mapped Netlist in Low Power in Encounter RTL Compiler](#)

Related commands:

[state_retention connect_power_gating_pins](#) on page 897

[state_retention swap](#) on page 898

state_retention connect_power_gating_pins

```
state_retention connect_power_gating_pins
```

Connects the power gating pins according to the driver specifications.

Use this command if you

- Replaced the sequential cells with state-retention cells after mapping and did not specify to hook up the power gating pins at that time (used `state_retention swap` command without the `-connect_power_gating_pins` option).
- Specified the driver specifications (`state_retention define_driver` commands) after mapping (although the mapping instructions were given before mapping)

Related Information

[State-Retention Cell Replacement when Starting with Mapped Netlist in Low Power in Encounter RTL Compiler](#)

Related attributes:

[power_gating_pin_class](#)
[power_gating_pin_phase](#)

state_retention swap

```
state_retention swap
  [-hierarchical]
  [-start_from instance]
  [-connect_power_gating_pins]
```

Replaces sequential cells with their equivalent state retention power gating cells in a mapped netlist.

Options and Arguments

-connect_power_gating_pins

Hooks up the power gating pins with their drivers.

The drivers are specified through the `lp_srpq_pg_driver` instance attribute.

-hierarchical

Allows traversing the design hierarchy to map.

By default, this command only affects instances at the current level of the design hierarchy.

-start_from *instance*

Starts replacing sequential cells from the specified hierarchical instance.

By default, the process starts from the current location in the design hierarchy.

Related Information

[State-Retention Cell Replacement when Starting with Mapped Netlist in Low Power in Encounter RTL Compiler](#)

Affected by these attributes:

[hdl_enable_proc_name](#)

[hdl_proc_name](#)

write_forward_saif

```
write_forward_saif
  [-library library_path...
   |-library_domain library_domain]
  [ > file ]
```

Prints the library forward SAIF file. This file contains the state-dependent and path-dependent (SDPD) directives needed to generate backward SAIF files during simulation.

Note: You do not need to have any designs loaded to write out a forward SAIF file. You only need to have the libraries loaded.

Options and Arguments

file Specifies the file to which to write the library forward SAIF information.

If not specified, the information is written to the screen.

-library library_path...

Specifies the paths to the libraries for which to generate the forward SAIF information.

If not specified, the information is generated for all libraries that are loaded.

-library_domain library_domain

Specifies the path to the library domain containing the libraries for which to generate the forward SAIF information.

If not specified, the information is generated for all libraries in all library domains.

Note: This option only applies if you created library domains using the [create_library_domain](#) command.

Examples

- The following example redirects the forward SAIF information for the cg library to the my.saif file:

```
write_forward_saif -library /libraries/cg > my.saif
```

Command Reference for Encounter RTL Compiler

Low Power Synthesis

- The following example redirects the forward SAIF information for the libraries in library domain d1 to the screen:

```
write_forward_saif -library_domain /libraries/library_domains/d1
```

Related Information

Related commands: [create_library_domain](#) on page 915
 [read_saif](#) on page 880

write_saif

```
write_saif [-duration simulation_period] [-computed]
           [-boundary_only] [-include_hier_ports] [> file]
```

Writes a hierarchical SAIF file containing the user-asserted or computed (if requested) probability and toggle count of the pins in the design.

By default, the RC-LP engine writes out the user-asserted switching activities of all leaf instance output pins and primary inputs ports.

The `write_saif` command writes out a compressed SAIF file if you add the `.gz` extension to the file name, but is not removed from the directory.

Options and Arguments

<code>-boundary_only</code>	Writes out the switching activities of the primary inputs ports and the leaf sequential instance output pins.
<code>-computed</code>	Adds the computed probability and toggle count of the pins and nets to the SAIF file. By default only the asserted values are written out.
<code>-duration <i>simulation_period</i></code>	Modifies the duration for which the toggle count is written in the SAIF file. By default, toggle counts are given for a duration of one second. By modifying the duration, smaller toggle counts (numbers) can be written. For example, if the toggle rate is <code>3e-3/ns</code> , the default printed toggle count is 300000. If the simulation period is set to <code>1e+5 ns</code> , the printed toggle count will be 300.
	Note: Do not choose the duration too small, otherwise the toggle count will be rounded to 0, because only integer numbers are written out.
	<i>Default:</i> <code>1e+9 ns</code>
<code>file</code>	Specifies the name of the file to which to write the probability values and toggle count values.
<code>-include_hier_ports</code>	Includes the (computed) switching activities for the hierarchical output ports.

Example

- The following example writes out a SAIF file with the user-asserted switching activities.

```
write_saif > my.saif
```

Note: If you did not read in a TCF or SAIF file, and you did not set toggle rate or probability values on any nets, this SAIF file will not contain any switching activities because you did not request to write out the computed values.

Related Information

Affected by these attributes: [lp_asserted_probability](#)

[lp_asserted_toggle_rate](#)

Related command: [read_saif](#) on page 880

By default, the command writes out the toggle count for a duration of 1s. For example, if the toggle rate is 3e-3/ns and the simulation period is 1e5 ns, the printed toggle count will be 300

write_tcf

```
write_tcf [-duration simulation_period] [-computed]
           [-hierarchical] [-include_hier_ports] [> file]
```

Writes a TCF file containing the user-asserted or computed (if requested) probability and toggle count of the pins in the design.

By default, the RC-LP engine writes out the user-asserted switching activities of all leaf instance output pins and primary inputs ports.

The `write_tcf` command writes out a compressed TCF file if you add the `.gz` extension to the file name.

Options and Arguments

<code>-boundary_only</code>	Writes out the switching activities of the primary inputs ports and the leaf sequential instance output pins.
<code>-computed</code>	Adds the computed probability and toggle count of the pins and nets to the TCF file. By default only the asserted values are written out.
<code>-duration <i>simulation_period</i></code>	Modifies the duration for which the toggle count is written in the TCF file. By default, toggle counts are given for a duration of 1s By modifying the duration, smaller toggle counts can be written. For example, if the toggle rate is 3e-3/ns, the default printed toggle count is 300000. If the simulation period is 1e5 ns, the printed toggle count will be 300. Note: Do not choose the period too small, otherwise the toggle count will be rounded to 0, because only integer numbers are written out. <i>Default:</i> 1e+9 ns
<code><i>file</i></code>	Specifies the name of the file to which to write the probability values and toggle count values.
<code>-hierarchy</code>	Writes out a TCF file in hierarchical format. By default, the RC-LP engine writes out a flat TCF file.

Command Reference for Encounter RTL Compiler

Low Power Synthesis

`-include_hier_ports`

Includes the (computed) switching activities for the hierarchical output ports.

Example

- The following example writes out a flat TCF file with the user-asserted switching activities.

```
write_tcf > my.tcf
```

Note: If you did not read in a TCF or SAIF file, and you did not set toggle rate or probability values on any nets, this TCF file will not contain any switching activities because you did not request to write out the computed values.

Related Information

[Troubleshooting in Low Power in Encounter RTL Compiler](#)

[TCF Syntax in Toggle Count Format Reference](#)

Related command: [read_tcf](#) on page 885

Affected by these attributes: [lp_asserted_probability](#)
[lp_asserted_toggle_rate](#)

Advanced Low Power Synthesis

- [check_cpf](#) on page 906
- [check_library](#) on page 908
- [commit_cpf](#) on page 914
- [create_library_domain](#) on page 915
- [isolation_cell_remove](#) on page 916
- [level_shifter_remove](#) on page 919
- [read_cpf](#) on page 922
- [reload_cpf](#) on page 924
- [report_isolation](#) on page 925
- [report_level_shifter](#) on page 926
- [verify_power_structure](#) on page 928
- [write_cpf](#) on page 930

check_cpf

```
check_cpf
  [-isolation | -level_shifter | -retention | -all]
  [-detail] [-continue_on_error]
  [-debug] [-run_dir directory] [-license string] [> file]
```

Checks the validity of the CPF rules against the RTL of the design. This enables designers to capture any violations of the low power intent of the design early in the design cycle.

- If no low power rule check errors are detected, the command returns 1.
- If rule check errors are detected, the command returns 0. In this case, you need to make the necessary changes to your CPF file before proceeding further with synthesis.

To run this command you need to have access to Encounter® Conformal® Low Power.

Options and Arguments

-all	Reports all problems with the CPF file. By default, all problems will be reported.
-debug	Creates a dofile and other required files for Encounter Conformal Low Power allowing you to debug any errors further in Conformal Low Power.
-continue_on_error	Allows the tool to continue when low power rule check errors are encountered.
-detail	Provides a detailed report.
file	Redirects the report to the specified file.
-isolation	Reports only problems with the isolation rules.
-level_shifter	Reports only problems with the level-shifter rules.
-license string	Specifies the Conformal license to be used for this command.
-retention	Reports only problems with the state retention rules.
-run_dir directory	Specifies the directory in which the required files for Encounter Conformal Low Power must be stored. <i>Default: .clp</i>

Examples

The following command reports problems with the level shifter rules.

```
rc:/designs/top> check_cpf -level_shifter
Using Conformal version xxx.

=====
CPF LEVEL SHIFTER VIOLATIONS
=====
CPF LS1: No level shifter rule specified for power domain crossing.
  Severity: Error      Occurrence: 8

Error   : Low Power rule check did not finish successfully. [RCLP-203] [check_cpf]
          : Fix the errors before proceeding further or set the attribute
'clp_treat_errors_as_warnings' appropriately.
0
```

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Check the CPF File](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Check the CPF File](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Check the CPF File](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

[MSV with DFT Flow in Design for Test in Encounter RTL Compiler](#)

[PSO with DFT Flow in Design for Test in Encounter RTL Compiler](#)

[Interfacing with Conformal Low Power in Interfacing between Encounter RTL Compiler and Encounter Conformal](#)

[Common Power Format Rule Checks in *Encounter® Conformal® Low Power Reference Manual*](#)

Related commands: [read_cpf](#) on page 922

Affected by this attribute: [wclp_lib_statetable](#)

check_library

```
check_library
    [-isolation_cell] [-level_shifter_cell]
    [-retention_cell]
    [-library_domain library_domain_list]
    [-libcell libcell_list]
    [> file]
```

Allows you to check specific information in the loaded libraries with regards to level shifters, isolation cells, and state retention cells. The report also lists the unusable cells. The information returned can be fine tuned by combining several options.

Without any options specified, this command lists the number of level shifters, isolation cells, and state retention cells available in each of the library domains. If no library domains exist, the report lists the library names instead.

Options and Arguments

<code>file</code>	Specifies the name of the file to which to write out the library information.
<code>-isolation_cell</code>	Returns two lists of library cells: <ul style="list-style-type: none">■ A list of pure isolation cells with their isolation type■ A list of combo cells with for each cell<ul style="list-style-type: none">□ The isolation type□ The voltage ranges that they support□ Whether the combo cell can be used between a lower and higher voltage, or vice versa□ The valid location for the cell
<code>-level_shifter_cell</code>	Returns a list of level shifter cells found in each of the library domains and specifies for each cell <ul style="list-style-type: none">■ The supported input and output range■ Whether the level shifter can be used between a lower and higher voltage, or vice versa■ The valid location for the cell

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

`-library_domain library_domain_list`

List the number of level shifters, isolation cells, and state retention cells available in each of the specified library domains.

`-libcell libcell_list`

If not combined with any other option, indicates for each of the specified library cells to which library domain it belongs, whether it is a level shifter, isolation cell, retention cell, always on cell, and the function of the cell.

`-state_retention_cell`

Returns for each library domain the following information:

- A list of sequential elements that have no state-retention equivalent
- A list of state-retention cells available in that domain

Examples

- The following command requests a general check of the libraries that were loaded. When requesting a general check, the report lists the number of usable and unusable level shifters, isolation cells, combo cells, and state retention cells in each library or library domain.

```
rc:/designs/Design2> check_library
=====
...
Module:           Design2
Library domain: lib_074v
  Domain index: 0
  Technology libraries: ....
  Operating conditions: nominal_ (balanced_tree)
Library domain: lib_090v_
  Domain index: 1
  Technology libraries: ....
  Operating conditions: nominal_ (balanced_tree)
Library domain: lib_110v_
  Domain index: 2
  Technology libraries: ....
  Operating conditions: nominal_ (balanced_tree)
Library domain: lib_120v_
  Domain index: 3
  Technology libraries: ...
  Operating conditions: nominal_ (balanced_tree)
Wireload mode:    enclosed
Area mode:       timing library
=====
```

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
lib_074v(0.74)	345	2	25	2	5
lib_120v(1.2)	341	0	13	0	5
lib_090v(0.9)	345	2	25	2	5
lib_110v(1.1)	343	2	17	2	5

Unusable libcells

Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
lib_074v(0.74)	249	34	45	16	49
lib_120v(1.2)	232	15	47	8	49
lib_090v(0.9)	249	34	45	16	49
lib_110v(1.1)	239	22	47	10	49

- The following command checks the libraries in library domain lib_120v.

```
rc:/> check_library -level_shifter_cell -library_domain lib_120v
=====
..... Library domain: lib_074v
          Domain index: 0
          Technology libraries: ...
          Operating conditions: BALANC_TREE (balanced_tree)
          Library domain: lib_120v
          Domain index: 1
          Technology libraries: ...
          Operating conditions: BALANC_TREE (balanced_tree)
          Wireload mode: enclosed
=====
```

Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
lib_120v(1.2)	11	3	10	2	0

Unusable libcells

Library Domain	Total cells	LS cell	ISO cell	Combo (LS+ISO)	SR Flops
lib_120v(1.2)	0	0	0	0	0

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

- The following command checks the libraries for isolation cells only. In the example below, the cell names in the Combo cell column were abbreviated for documentation purposes to fit the report. The cell names are not abbreviated by the tool.

```
rc:/> check_library -isolation_cell
=====
.....  
Library domain: lib_074v  
Domain index: 0  
Technology libraries: ...  
Operating conditions: BALANC_TREE (balanced_tree)  
Library domain: lib_120v  
Domain index: 1  
Technology libraries: ...  
Operating conditions: BALANC_TREE (balanced_tree)  
Wireload mode: enclosed
=====  
  
Pure isolation cells  
=====  
=====  
Library Domain Isolation Type Isolation cells  
-----  
lib_074v enable_high_out_high OR2XCP  
OR2XHP  
....  
AND2XCP  
AND2XH  
....  
lib_120v enable_high_out_high OR2XCPPP  
OR2XHPPP  
....  
AND2XCPPP  
AND2XHPPP  
....  
-----  
  
Combo cells  
=====  
=====  
Library Isolation Combo Input Output Direction Location  
Domain type cell Range (V) Range (V)  
-----  
lib_074v enable_high_out_high .....BH1XH 0.74-0.74 1.2-1.2 up to  
.....BH1XL 0.74-0.74 1.2-1.2 up to  
enable_low_out_low .....1CLXH 1.2-1.2 0.74-0.74 down to  
.....1CLXL 1.2-1.2 0.74-0.74 down to  
.....PPPAD 0.74-0.74 1.2-1.2 up to  
.....PPPAD 0.74-0.74 1.2-1.2 up to  
lib_120v enable_high_out_high .....BH1XH 0.9-0.9 1.2-1.2 up to  
enable_low_out_low .....1CLXH 0.9-0.9 1.2-1.2 up to
-----
```

Note: If the libraries would have unusable isolation cells, the report would list the names of the isolation cells per library or library domain.

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

- The following command checks the function of library cell LSHLL1CLXL.

```
rc:/> check_library -libcell LSHLL1CLXL
=====
 Library      Libcell      Level   Isolation   Retention   Always   Function
 domain       domain       shifter    cell        flop        ON
 -----
 lib_074v     LSHLL1CLXL  true     true       false      false    Y = A * B
```

- The following command checks the level shifter characteristics of the specified cell.

```
rc:/> check_library -libcell LSHLL1CLXL -level_shifter_cell
=====
 .....
 Library domain:          lib_074v
 Domain index:            0
 Technology libraries: ...
 Operating conditions: BALANC_TREE (balanced_tree)
 Library domain:          lib_120v
 Domain index:            1
 Technology libraries: ...
 Operating conditions: BALANC_TREE (balanced_tree)
 Wireload mode:           enclosed
 =====

 Level shifter report
 =====

=====
 Library      Level shifter      Input      Output      Direction   Location
 Domain       cell             Range (V)   Range (V)
 -----
 lib_074v     LSHLL1CLXL     1.2-1.2    0.74-0.74  down       to
```

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

- The following command checks the libraries for state retention cells. The report distinguishes between flip-flops and latches. In the example below the library has no latches.

```
rc:/> check_library -retention_cell
=====
.....
=====

Flops without corresponding state-retention flops
-----

=====

Library Domain          Flops
-----
110_lib      HD65_LS_DFPHQNX5      HD65_LS_DFPRQNX10      HD65_LS_SDFNRX5
              HD65_LS_SDFPHQNX10     HD65_LS_SDFPSQNX20
-----
Usable state-retention flops
-----

=====

Library Domain          Flops
-----
090_lib      HD65_LS_SDFNRX10_SRPG  HD65_LS_SDFNRX5_SRPG
110_lib      HD65_LS_SDFPHQNX10_SRPG  HD65_LS_SDFPHQX10_SRPG  HD65_LS_SDFPQNX10_SRPG
              HD65_LS_SDFPQX10_SRPG  HD65_LS_SDFPRQNX10_SRPG  HD65_LS_SDFPRQX10_SRPG
              HD65_LS_SDFFRSQX10_SRPG  HD65_LS_SDFPSQNX10_SRPG  HD65_LS_SDFPSQX10_SRPG
-----
Latches without corresponding state-retention latches
-----

=====

Library Domain          Latches
-----
Usable state-retention latches
-----

=====

Library Domain          Latches
-----
```

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Read Target Libraries](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Read Target Libraries](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Read Target Libraries](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

Related commands: [read_cpf](#) on page 922

commit_cpf

```
commit_cpf  
  [-level_shifter_only]  
  [-isolation_cell_only]
```

Inserts level-shifter logic and isolation logic as requested based on the rules specified in previously read in CPF file(s). If specified without any options, both level-shifter logic and isolation logic are inserted.

Options and Arguments

<code>-level_shifter_only</code>	Inserts only level-shifter logic according the rules specified in CPF file(s).
<code>-isolation_cell_only</code>	Inserts only isolation logic.

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Execute CPF File](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Execute CPF File](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Execute CPF File](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

[MSV with DFT Flow in Design for Test in Encounter RTL Compiler](#)

[PSO with DFT Flow in Design for Test in Encounter RTL Compiler](#)

Related commands:

<u>read_cpf</u> on page 922
<u>reload_cpf</u> on page 924

create_library_domain

```
create_library_domain domain_list
```

Creates the specified library domains. To use dedicated libraries with portions of the design, you must use this command before you read in any libraries for the specified library domains. The command returns the directory path to the library domains that it creates.

You can find the objects created by the `create_library_domain` command in:

```
/libraries/library_domains
```

Note: There is no limitation on the number of library domains you can create.

Options and Arguments

<i>domain_list</i>	Specifies the names of the library domains to be created. Specify the library domains as a Tcl list.
--------------------	---

Examples

- The following example creates three library domains:

```
rc:/> create_library_domain {dom_1 dom_2 dom_3}
/libraries/library_domains/dom_1 /libraries/library_domains/dom_2 /libraries/
library_domains/dom_3
```

Related Information

[Create Library Domains in *Encounter RTL Compiler Synthesis Flows*.](#)

Related attribute: [library](#)

isolation_cell remove

```
isolation_cell remove
[ iso_hier_instance_list | -hierarchical ]
[-from_power_domain power_domain_list]
[-to_power_domain power_domain_list]
```

Removes hierarchical isolation cell instances from the design or the current hierarchical instance. When you combine options only those hierarchical isolation cell instances that meet all criteria are removed. The command returns the number of (hierarchical and leaf) isolation cell instances that were removed.

RTL Compiler can remove any isolation logic that was *inserted* by RTL Compiler.

If the isolation logic was *imported*, RTL Compiler can only remove the hierarchical instance if the following conditions are met:

1. The individual cells in the hierarchy are *all*
 - a. Mapped
 - b. Of the same type
 - Pure isolation cells
 - Pure isolation cells with at most one inverter at their enable
 - *Combo* cells that combine level shifting and isolation
 - Discrete cells—AND or OR cells with or without an inverter at one of the inputs.
2. The enable, data and out pins of each leaf isolation cell contained in isolation hierarchy can be clearly distinguished.



This command does not remove any ports that were created during isolation logic insertion.

Options and Arguments

-from_power_domain power_domain_list

Removes all isolation cell instances whose drivers are output pins of instances in the specified power domains.

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

-hierarchical Removes all isolation cell instances (that meet the from and to criteria) down the hierarchy starting from the current directory.

iso_hier_instance_list

Specifies the names of the hierarchical isolation cell instances to be removed.

-to_power_domain *power_domain_list*

Removes all isolation cell instances whose output pins are driving instances in the specified power domains.

Examples

- The following command removes all isolation cell instances that connect to the top-level of the design.

```
rc:/> isolation_cell remove -hier
Removing isolation cells from '/designs/top' ...
Status
=====
Number of isolation cell hierachical instances removed: 8
Isolation cells removed: 8
8
```

- The following command removes all isolation cell instances driving instances in power domain p4.

```
rc:/> isolation_cell remove -to_power_domain p4
Removing isolation cells from '/designs/top' ...
Status
=====
Number of isolation cell hierachical instances removed: 3
Isolation cells removed: 3
3
```

- The following command removes two specific isolation cell instances.

```
rc:/> isolation_cell remove {mux_10_14/RC_ISO_HIER_INST_23 \
 mux_10_14/RC_ISO_HIER_INST_24}
Removing isolation cells from '/designs/top' ...
Status
=====
Number of isolation cell hierachical instances removed: 2
Isolation cells removed: 2
2
```

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

- The following command removes isolation cell instances that are driven by instances in power domain p1 and that are driving instances in both power domains p2 and p4.

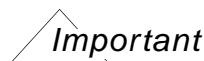
```
rc:/> isolation_cell remove -from_power_domain p1 -to_power_domain {p2 p4}
Removing isolation cells from '/designs7top' ...
Status
=====
Number of isolation cell hierachical instances removed: 1
Isolation cells removed: 1
1
```

level_shifter remove

```
level_shifter remove
{ [instance]...
| [-from_library_domain library_domain...]
[-to_library_domain library_domain...]
[-instance_from instance_list]
[-instance_to instance_list]
[-hierarchical]
[-invalid_only]
```

Removes leaf or hierarchical level-shifter instances. Without any options specified, the command can remove level shifters from the design or the current hierarchical instance. The command returns the number of leaf level-shifter instances that were removed.

The RC-LP engine can remove invalid *imported* level shifters only if you specify the `-invalid_only` option and the output of the imported level shifter is unconnected.



If the `preserve` attribute on a leaf level shifter or its module is set to `true`, or `size_ok`, or `map_size_ok`, it will not be removed. It can only be removed if the `preserve` attribute was set to `delete_ok`, `size_delete_ok`, or `false`.

Options and Arguments

`-from_library_domain library_domain`

Removes all level shifters whose drivers are output pins of instances in the specified library domain.

`-hierarchical`

Removes all level shifters down the hierarchy starting from the current directory.

`instance`

Specifies the name of a leaf or hierarchical level-shifter instance to be removed.

`-instance_from instance_list`

Removes all level shifters whose input pin is connected to

- a specified leaf instance
- an instance that is an immediate child of a specified hierarchical instance.

`-instance_to instance_list`

Removes all level shifters whose output pin is connected to

- a specified leaf instance
- an instance that is an immediate child of a specified hierarchical instance.

`-invalid_only` Removes only level shifters that have become invalid.

`-to_library_domain library_domain`

Removes all level shifters whose output pins are driving instances in the specified library domain.

Examples

- The following example removes all individual level shifters that connect to the top-level of the design.

```
rc:/> level_shifter remove
Info      : Total level shifter hierarchical instances removed. [LS-102]
           : 1
Info      : Total level shifter cells removed. [LS-103]
           : 12
12
```

- The following example removes all individual level shifters in the design across the hierarchy.

```
rc:/designs/top> level_shifter remove -hier
level_shifter remove: remove level shifters
Info      : Total level shifter hierarchical instances removed. [LS-102]
           : 2
Info      : Total level shifter cells removed. [LS-103]
           : 13
13
```

- The following example removes the level-shifter instance RC_LS_HIER_INST_56.

```
rc:/> level_shifter remove [find / -inst RC_LS_HIER_INST_56]
Info      : Total level shifter hierarchical instances removed. [LS-102]
           : 1
Info      : Total level shifter cells removed. [LS-103]
           : 12
12
```

Note: You can find the list of hierarchical level-shifter instance names in a detailed hierarchical level-shifter report.

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

- The following example removes all level shifters that have drivers in library domain 07v.

```
rc:/> level_shifter remove -from [find / -library_domain 07v]
Info      : Total level shifter hierarchical instances removed. [LS-102]
           : 1
Info      : Total level shifter cells removed. [LS-103]
           : 12
```

- The following example removes all individual level shifters that connect to instance outa_reg[6].

```
rc:/> level_shifter remove -instance_to [find / -inst outa_reg[6]]
Info      : Total level shifter hierarchical instances removed. [LS-102]
           : 0
Info      : Total level shifter cells removed. [LS-103]
           : 1
```

Related Information

Related commands: [report level_shifter](#) on page 926

read_cpf

```
read_cpf [-library] [-design design]  
[-read_sdc_options string...] file [file]...
```

Reads the power intent for the design from the specified Common Power Format (CPF) file(s).

Note: In general the `read_cpf` command does not change the netlist. However, the netlist is changed if the CPF file contains

- A `set_instance` command specified with the `-port_mapping` option
The `-port_mapping` option specifies the mapping of the virtual ports.
- A `set_design` command specified with the `-ports` option
The `-ports` option specifies a list of virtual ports. These ports do not exist in the definition of the module but will be needed for the control signals of the low power logic such as isolation logic, state-retention logic, and so on.

Options and Arguments

<code>-design <i>design</i></code>	Specifies the design for which the CPF files are read in. This option is only required when multiple designs are loaded in the session.
<code><i>file</i></code>	Specifies the name of the CPF file. The file can have any name, suffix, or length.
<code>-library</code>	Instructs to process the following statements in the CPF file: <code>define_isolation_cell</code> <code>define_level_shifter_cell</code> <code>define_library_set</code> <code>define_state_retention_cell</code>
<code>-read_sdc_options <i>string</i></code>	Allows you to specify the options that you would normally specify for the <code>read_sdc</code> command in a non-CPF flow. Note: The <code>-mode</code> option is not supported with <code>-read_sdc_options</code> .

Example

```
read_cpf -read_sdc_options "-stop_on_errors" mycpf.cpf
```

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Read CPF File](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Read CPF File](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Read CPF File](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

[MSV with DFT Flow](#) in *Design for Test in Encounter RTL Compiler*

[PSO with DFT Flow](#) in *Design for Test in Encounter RTL Compiler*

Related commands:

[commit_cpf](#) on page 914

[reload_cpf](#) on page 924

reload_cpf

```
reload_cpf  
[-design design]
```

Applies previously loaded constraints as needed to new design objects (ports/pins/nets/instances) that are created during synthesis.

Note: This command is only required if you use CPF 1.0.

Options and Arguments

-design <i>design</i>	Specifies the design for which you want to reapply the previously loaded constraints. If you omit this option, the constraints will be reapplied to the current design.
------------------------------	--

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Reload CPF File](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Reload CPF File](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Reload CPF File](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

Related commands:

commit_cpf on page 914	read_cpf on page 922
--	--------------------------------------

report isolation

For more information, refer to [report isolation](#) in Chapter 8, “Analysis and Report.”

report level_shifter

For more information, refer to [report level_shifter](#) in [Chapter 8, “Analysis and Report.”](#)

report state_retention

For more information, refer to [report state_retention](#) in [Chapter 8, “Analysis and Report.”](#)

verify_power_structure

```
verify_power_structure
  [-isolation] [-level_shifter] [-retention] [-all]
  {-pre_synthesis | -post_synthesis} [-detail]
  [-run_dir directory] [-debug]
  [continue_on_error] [-license string] [> file]
```

Verifies whether the low power cells in the design conform to the rules and specifications in the loaded CPF file. Specifically, RTL Compiler will flag if there are any missing isolation, level shifter, or state retention cells or if the low power cells are not connected appropriately.

To run this command you need to have access to Encounter® Conformal® Low Power.

Options and Arguments

<code>-all</code>	Reports all violations.
<code>-continue_on_error</code>	Allows the tool to continue when low power rule check errors are encountered.
<code>-detail</code>	Provides a detailed report.
<code>-debug</code>	Creates a dofile and other required files for Encounter Conformal Low Power allowing you to debug any errors further in Conformal Low Power.
<code>file</code>	Redirects the report to the specified file.
<code>-isolation</code>	Reports only isolation related violations.
<code>-level_shifter</code>	Reports only level-shifter related violations.
<code>-license string</code>	Specifies the Conformal license to be used for this command.
<code>-post_synthesis</code>	Runs Conformal Low Power, after synthesis, on low power cells.
<code>-pre_synthesis</code>	Runs Conformal Low Power, before synthesis, to check the existing low power cells.
<code>-retention</code>	Reports only state retention related violations.
<code>-run_dir directory</code>	Specifies the directory in which the required files for Encounter Conformal Low Power must be stored. <i>Default:</i> .

Command Reference for Encounter RTL Compiler

Advanced Low Power Synthesis

Related Information

See the following sections in *Low Power in Encounter RTL Compiler*

- [Verify Added Power Logic](#) in “Using CPF for Multiple Supply Voltage Designs”
- [Verify Added Power Logic](#) in “Using CPF for Designs Using Power Shutoff Methodology”
- [Verify Added Power Logic](#) in “Using CPF for Designs Using Dynamic Voltage Frequency Scaling”

[MSV with DFT Flow in Design for Test in Encounter RTL Compiler](#)

[PSO with DFT Flow in Design for Test in Encounter RTL Compiler](#)

[Interfacing with Conformal Low Power in Interfacing between Encounter RTL Compiler and Encounter Conformal](#)

Related commands: [commit_cpf](#) on page 914

[reload_cpf](#) on page 924

Affected by this attribute: [wclp_lib_statetable](#)

write_cpf

```
write_cpf
  [-design design]
  [-separate_library_cpf | -use_library_cpf file]
  [-no_sdc_update
  | [-write_sdc_options string] [-honor_cpf_sdc] ]
  [-output_directory string]
  [-prefix string] [-overwrite]
```

Writes out an updated CPF file. The command returns the path to the file with the design CPF information.

The golden (original) CPF file references design objects in the RTL or original netlist. During synthesis, changes to instance and pin names can occur. The new CPF file contains the power intent of the golden CPF file but with updated references to the design objects.

Options and Arguments

-design <i>design</i>	Specifies the design for which to write out the CPF file(s).
-honor_cpf_sdc	Writes out SDC files for those power modes for which SDC files were specified in the CPF file through the update_power_mode command. By default, the write_cpf command writes out the SDC files for all modes. You cannot specify this option with the -no_sdc_update option.
-no_sdc_update	Prevents to write out updated SDC files.
-output_directory <i>string</i>	Specifies the directory to which the CPF file(s) must be written.
-overwrite	Allows to overwrite any existing files.
-prefix <i>string</i>	Specifies the prefix for the CPF files. The prefix can contain letters (of the alphabet), digits, and the underscore (_). That is, you cannot include any of the following characters: () [] { } \$ \ /
-separate_library_cpf	Writes the library information in a separate CPF file.

`-use_library_cpf file`

Prevents the library information from being written out and includes the specified file as the library CPF file.

`-write_sdc_options string`

Specifies extra options to apply when writing out the SDC file(s).

You cannot specify this option with the `-no_sdc_update` option.

Examples

- The following command specifies how to write out the SDC files.

```
rc:/designs/Design> write_cpf -output_directory final \
    -write_sdc_options "-strict -exclude {set_clock_latency}"
```

- The following command writes out the CPF files for design Design.

```
rc:/designs/Design> write_cpf -prefix mycpf_ -output_directory final \
    -separate_library_cpf
Written library information in working_directory/final/mycpf_Design.lib.cpf.
Written design cpf information in working_directory/final/mycpf_Design.cpf.
working_directory/final/mycpf_Design.cpf
```

Related Information

Related command: [read_cpf](#) on page 922

Command Reference for Encounter RTL Compiler
Advanced Low Power Synthesis

Design Manipulation

- [change_link](#) on page 935
- [change_names](#) on page 937
- [clock_gating](#) on page 944
- [delete_unloaded_undriven](#) on page 945
- [edit_netlist](#) on page 946
- [edit_netlist_bitblast_all_ports](#) on page 948
- [edit_netlist_bitblast_port](#) on page 949
- [edit_netlist_connect](#) on page 950
- [edit_netlist_dedicate_subdesign](#) on page 952
- [edit_netlist_delete](#) on page 953
- [edit_netlist_disconnect](#) on page 954
- [edit_netlist_group](#) on page 956
- [edit_netlist_hier_connect](#) on page 958
- [edit_netlist_new_design](#) on page 959
- [edit_netlist_new_instance](#) on page 960
- [edit_netlist_new_port_bus](#) on page 962
- [edit_netlist_new_primitive](#) on page 963
- [edit_netlist_new_subport_bus](#)
- [edit_netlist_ungroup](#) on page 966
- [edit_netlist_uniquify](#) on page 967
- [group](#)
- [insert_tiehilo_cells](#) on page 969

Command Reference for Encounter RTL Compiler

Design Manipulation

- [mv on page 972](#)
- [remove_cdn_loop_breaker on page 974](#)
- [reset_design on page 976](#)
- [rm on page 977](#)
- [ungroup on page 978](#)
- [uniquify](#)

change_link

```
change_link -instances instance_list
  { -design_name {instance | subdesign | design}
  | -libcell libcell }
  [-pin_map string] [-lenient]
  [-no_name_change]
  [-change_in_non_uniq_subd]
  [-copy_attributes] [-retain_exceptions]
```

Changes the reference of a hierarchical instance to the specified subdesign or design. The command also supports libcell to libcell reference changes as well as a hierarchical instance to libcell changes.

Options and Arguments

-change_in_non_uniq_subd	Changes the link of the instance(s) in all instantiations of a non-uniquified subdesign.
-copy_attributes	Copies the attributes of the original leaf instance(s) to the new leaf instances.
-design_name { <i>instance</i> <i>subdesign</i> <i>design</i> }	Specifies the design, subdesign, or hierarchical instance to which the link has to be changed.
-instances <i>instance_list</i>	Specifies the instance(s) whose reference has to be changed.
-lenient	Leaves the pins floating if a pin map is not found.
-libcell <i>libcell</i>	Specifies the library cell to which the instance link has to be changed.
-no_name_change	Prevents renaming of the subdesign name.
-pin_map <i>string</i>	Specifies, as a Tcl list of lists, the required pin mapping for swapping.
-retain_exceptions	Keeps the original exceptions of the instance after replacing it with another link.

Examples

- The following command changes the reference of the hierarchical instances `top/A`, `top/B`, and `top/C` to `patch`.

```
change_link -instances {/designs/top/instances_hier/A \
/designs/top/instances_hier/B /designs/top/instances_hier/C} \
-design_name /designs/patch
```

- The following example changes the reference of the hierarchical instance `add_0` to the design `add`:

```
rc:/> change_link -design_name add \
-instance /designs/test/instances_hier/add_0

CHLINK INFO : Changing link of instance /designs/test/instances_hier/add_0 to
design /designs/add
Warning : Uniquifying instance /designs/test/instances_hier/add_0. New
subdesign is add_1
```

- In the following example, `A1`, `A2`, and `A3` are instances of subdesign `A` which is not unqualified. The following command changes a leaf instance in all instances of subdesign `A`.

```
change_link -instances {/designs/top/instances_hier/A1/instances_comb/g1} \
-libcell buf1 -change_in_non_uniq_subd
```

This command changes not only leaf instance `A1/g1` but also `A2/g1` and `A3/g1` with `buf1`.

Note: If you omit the `-change_in_non_uniq_subd` option, the tool will issue an error.

- The following command replaces hierarchical instance `A1` with `patch` and tries to retain all exceptions of `A1`.

```
change_link -instances {/designs/top/instances_hier/A1} \
-design_name /designs/patch -retain_exceptions
```

Exceptions defined for instance `A1` are copied to `patch` if the object for which the exception was originally defined is also found in `patch`. For example, an exception defined on `A1/d_reg` will be retained if `d_reg` also exists in `patch`.

- The following command specifies how to map the pins of the hierarchical instance `A` to the pins of design `new`. Instance `A` has pins `a`, `b`, `c`, and `d`. Design `new` has pins `e`, `f`, `g`, and `h`.

```
change_link -instances /designs/top/instances_hier/A \
-design_name /designs/new -pin_map {{a e} {b f} {c g} {d h}}
```

- The following commands replace hierarchical instance `U1` with top-level design `digit` and hierarchical instance `U2` with top-level design `digit_1` while retaining the original top-level design names as the subdesign names.

```
change_link -instances /des*/top/instances_hier/U1 -design_name /designs/digit
change_link -instances /des*/top/instances_hier/U2 -design_name /des*/digit_1
```

change_names

```
change_names [ -net | -instance | -design | -subdesign  
| -port_bus | -subport_bus]...  
[-local] [-force] [-lec]  
[-vhdl] [-verilog] [-system_verilog]  
[-prefix string [-name_collision]]  
[-suffix string [-name_collision]]  
[-first_restricted string] [-restricted string]  
[-last_restricted string] [-replace_str string]  
[-reserved_words string] [-max_length number]  
[-map string] [-allowed string... [-regexp]]  
[-check_internal_net_name] [-collapse_name_space]  
[-dummy_net_prefix string]  
[-case_insensitive] [-lowertoupper] [-uppertolower]  
[-log_changes file [-append_log]] [hier_instance]
```

Changes names of nets, busses, instances, designs, subdesigns, ports, port buses, and subport buses. You can specify one or more object types. If no object type is specified, the requested change applies to all object types unless otherwise specified. There is no restriction on the length of the name.

By default, all changes are global: changes are made to all objects (instances of the specified object types) in the design. You can specify the name of a hierarchical instance to restrict the changes to only the objects in that instance.

To change the name of a single object (net, instance, and so on), use the `mv` command.

Options and Arguments

`-allowed string` Specifies the characters that are allowed in names. Any characters that are not in the allowed list will be ignored in the resulting names. The minimum specification is 10 characters. To allow all the letters from a to z in capital and lower case letters, you must specify all of them. That is, you cannot use a dash (“-”) to indicate inclusion.

`-append_log` Appends the information of the last `change_names` command to the logfile specified with the `-log_changes` option

If you omit this option, the information of the last `change_names` command overwrites the current information in the logfile.

Note: You can only specify this option if you specified the `-log_changes` option.

Command Reference for Encounter RTL Compiler

Design Manipulation

-case_insensitive Does not take case sensitivity into account.

-check_internal_net_name

Adds the suffix `_int` to any net whose name matches that of a port or subport but is not connected to that port or subport.

-collapse_name_space

Adds the suffix `_design` to either the port, subport, net, or subdesign in a hierarchy only if they have similar Verilog names.

-design

Changes the names of design objects.

-dummy_net_prefix *string*

Specifies the prefix to use for the names of dummy nets when writing out the netlist or HDL.

Note: This option does not change the names in the design hierarchy.

-first_restricted *string*

Specifies the characters that cannot be used as first character in a name.

-force

Forces the name change even if the object name is preserved.

hier_instance

Specifies the name of the hierarchical instance to which the changes must be applied.

-instance

Changes the names of instance objects.

-last_restricted *string*

Specifies the characters that cannot be used as last character in a name.

-lec

Captures the changes in the log file in LEC preferred format.

-local

Restricts the changes to the current directory.

Default: global changes

-log_changes *file*

Specifies the name of the logfile that shows which names were changed using the `change_names` command and the result of the changes.

Command Reference for Encounter RTL Compiler

Design Manipulation

-lowerToUpper	Changes the names of all objects from lowercase to uppercase. This applies to objects of type instance, port, net, subport, design, and subdesign.
-map { { "from" "to" } . . . }	Maps the specified <i>from</i> character to the specified <i>to</i> character. Enclose each character in double quotes and separate the characters with a space. Enclose each set in braces. If you specify several sets, separate them with spaces and enclose the list of all sets with braces.
-max_length integer	Limits the length of the changed name to the specified number. If the resulting names are longer than the specified integer, the last letters will be truncated.
-name_collision	Indicates to only use the specified prefix or suffix values to change object names if a name clash would occur while executing the <code>change_names</code> command using other options.
-net	Changes the names of net objects.
-port_bus	Changes the names of the top-level port bus objects. Note: You cannot change the left bracket, "[", and the right bracket, "]", because they are a part of the bus name when referencing individual bits of the bus.
-prefix <i>string</i>	Adds a prefix to the names of the objects to be changed.
-regexp	Allows you to specify character ranges with the -allowed option.
-replace_str <i>string</i>	Specifies the replacement string. Specify NULL for a null string. <i>Default:</i> _
-reserved_words <i>string</i>	Specifies words to be avoided, such as "begin end".
-restricted <i>string</i>	Specifies the list of strings that cannot be used in a name. Separate independent patterns to be replaced with a "space." These strings are replaced with the -replace_str string.

Command Reference for Encounter RTL Compiler

Design Manipulation

-subdesign	Changes the names of subdesign (object).
-subport_bus	Changes the names of subport bus objects. Note: You cannot change the left bracket, “[”, and the right bracket, “]”, because they are a part of the bus name when referencing individual bits of the bus.
-suffix	Adds a suffix to the names of the objects to be changed.
-system_verilog	Replaces SystemVerilog reserved words.
-uppertolower	Changes the names of all objects from uppercase to lowercase. This applies to objects of type instance, port, net, subport, design, and subdesign.
-verilog	Replaces Verilog reserved words.
-vhdl	Replaces VHDL reserved words as well as strings that start with a digit, an underscore, continuous (two or more) underscores, and end with an underscore. You do not need to specify the -case_insensitive, -instance, or -subdesign option if you specify the -vhdl option.

Examples

- The following example adds a suffix `_t` to the design object names:

```
rc:/>change_names -design -suffix _t
```

All design objects will now have the `_t` suffix:

```
module top_newName (
    mid m1_t (....)
    mid1 m2_t (....)
    mid3 m3_t (....)
    INVXL g5_t
)
endmodule

module mid (...)
    out_reg_7_t (....)
)
endmodule
```

- The following example replaces all lowercase “n” with uppercase “N”, and all underscores “_” with hyphens “-” in all instance names.

```
rc:/> change_names -instance -map {{"n" "N"} {"_" "-"} }
```

- The following example replaces all lowercase “a” with uppercase “A” on all subdesigns and subports.

```
rc:/> change_names -map {{"a" "A"} } -subdesign -subport_bus
```

Command Reference for Encounter RTL Compiler

Design Manipulation

- The following example replaces any instances of ab, bc, or ca with "@" in all object names.

```
rc:/> change_names -restricted "ab bc ca" -replace_str "@"
```

- The following example specifies the maximum length of all subdesign names to be 12 characters. Issue this command after elaboration or before writing out the netlist.

```
rc:/> change_names -max_length 12 -subdesign
```

- The following example ignores case sensitivity. Because the design contains nets n_73 and N_73, RTL Compiler renames one of the nets to avoid a naming conflict.

```
rc:/> mv n_73 N_73  
/designs/alu/nets/N_73
```

```
rc:/> mv n_72 n_73  
/designs/alu/nets/n_73
```

```
rc:/> change_names -case_insensitive -net  
Info      : Change names is successful [CHNM-102]  
           : /designs/alu/nets/n_73 moved to /designs/alu/nets/n_73_1
```

- The following example illustrates that you cannot change the brackets ("[" and "]") when they are a part of the bus name referencing individual bits of the bus:

```
rc:/designs/test/ports_in> ls  
.          SI2      clk1      in1[0]     in2[0]     in2[3]     in3[2]     in3[5]  
rc:/designs/test/ports_in> change_names -port_bus -map {[ "[" "(" } {" "]" ")" }\n"}  
rc:/designs/test/ports_in> ls  
.          SI2      clk1      in1[0]     in2[0]     in2[3]     in3[2]     in3[5]
```

- The following two commands both change the brackets ("[" and "]") in the instance name to "x"s:

```
change_names -instance -restricted {[ ]} -replace_str "x"  
change_names -instance -restricted "\[ \]" -replace_str "x"
```

Note: There is no need to escape special characters when enclosing them in braces ({}). If you added the escape character inside the braces, the tool would try to replace this character as well with "x".

- The following example allows all capital and lower case letters, numbers, underscores, backslashes, and brackets:

```
rc:/> change_names -allowed \  
ABCDEFGHIJKLMNPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_\[\]
```

Note: You cannot use the dash "-" to indicate inclusion. That is, the following example is not allowed:

```
rc:/> change_names -allowed a-zA-Z0-9_[]
```

- The following example creates a separate `change_names` log file, `danni.log`, that reflects the subdesign name change:

```
rc:/> change_names -map {"SUB" "HERO_SUB"} -subdesign -log_changes danni.log
```

Command Reference for Encounter RTL Compiler

Design Manipulation

- The following example indicates that the `d` cannot be used as the first character in a net name and that when a name collision would occur, the prefix `xx` can be used.

```
change_names -name_collision -first_res d -prefix xx
```

- The following example shows a part of a netlist that includes VHDL reserved words:

```
generate u1(.A (eg1[0] ), .B (B[0]), .Y (Y[0]));  
open u2(.A (eg1[1] ), .B (B[1]), .Y (Y[1]));  
endmodule u3(.A (eg1[2] ), .B (B[2]), .Y (Y[2]));
```

To change, or eliminate, the names of the VHDL reserved words, use the `-vhdl` option.

```
rc:/> change_names -vhdl
```

Now the netlist does not contain any VHDL keywords:

```
generate_cn u1(.A (eg1[0] ), .B (B[0]), .Y (Y[0]));  
open_cn u2(.A (eg1[1] ), .B (B[1]), .Y (Y[1]));  
endmodule u3(.A (eg1[2] ), .B (B[2]), .Y (Y[2]));
```

- The following example changes all the object names from lowercase to uppercase:

```
rc:/> ls /designs/test/ports_in  
. in1[0] in1[1] in1[2] in1[3] in2[0] in2[1] in2[2]  
rc:/> change_names -lowertoupper  
Setting in1[3] --> IN1[3]  
Setting in1[2] --> IN1[2]  
Setting in1[1] --> IN1[1]  
...  
rc:/> ls /designs/test/ports_in  
. IN1[0] IN1[1] IN1[2] IN1[3] IN2[0] IN2[1]
```

Notice how the lowercase design name change to uppercase as well. The `-uppertolower` option works similarly, except that it changes all uppercase letters to lowercase.

- The following example specifies the character ranges that are allowed when renaming instances.

```
change_names -regexp -allowed "a-zA-Z0-9" -instance
```

- The following example uses the `MY_UNCONN_` prefix for dummy nets when writing out the netlist.

```
write_hdl  
change_names -dummy_net_prefix MY_UNCONN_  
write_hdl
```

Netlist before net name changes:

```
wire [1:0] in1, in2, in3;  
wire [5:0] out1, out2;  
wire UNCONNECTED, n_0, t;  
assign out2[3] = 1'b0;  
assign out2[4] = 1'b0;  
assign out2[5] = 1'b0;
```

Command Reference for Encounter RTL Compiler

Design Manipulation

```
assign out1[1] = 1'b0;
a a_1(.in1 ({in1[1], 1'b0}), .in2 ({in2[1], 1'b0}), .out1
({out1[5:2], UNCONNECTED, out1[0]}));
```

Netlist after the net name changes:

```
wire \MY_UNCONN_, n_0, t;
assign out2[3] = 1'b0;
assign out2[4] = 1'b0;
assign out2[5] = 1'b0;
assign out1[1] = 1'b0;
a a_1(.in1 ({in1[1], 1'b0}), .in2 ({in2[1], 1'b0}), .out1
({out1[5:2], \MY_UNCONN_, out1[0]}));
```

- The following example specifies that inside hierarchical instance m2, any changed names cannot end with the character 2.

```
change_names -last_restricted 2 /designs/m1/instances_hier/m2
```

Related Information

Related command: [write_hdl](#) on page 255

clock_gating

Refer to [clock_gating](#) in [Chapter 11, “Low Power Synthesis.”](#)

delete_unloaded_undriven

```
delete_unloaded_undriven  
  [-disconnect] [-force_bit_blast] [-all]  
  [design] [> file]
```

Disconnects subports and hierarchical pins connected to constants and that do not fanout to anything, and deletes unloaded and undriven subports from the design. Use this command as a post-processing step to remove any unused subports from the netlist.

By default the command skips individual bits of a bus that are connected to constants or that are unused.

Options and Arguments

<i>-all</i>	Disconnects subports and hierarchical pins connected to constants and that do not fanout to anything, and deletes unloaded and undriven subports and top-level ports from the design.
<i>design</i>	Specifies the design from which to remove the unused ports or subports.
<i>-disconnect</i>	Only disconnects the subports and hierarchical pins that are connected to constants and that do not fanout to anything.
<i>file</i>	Specifies the name of the file to which the output of the command should be redirected.
<i>-force_bit_blast</i>	Bitblasts modules that have individual bus bits that are unused or connected to constants and deletes the unused bus bits.
<i>-verbose</i>	Enables verbose output.

Command Reference for Encounter RTL Compiler

Design Manipulation

edit_netlist

```
edit_netlist { bitblast_all_ports | connect  
| dedicate_subdesign | disconnect  
| group | new_design | new_instance  
| new_port_bus | new_primitive | new_subport_bus  
| ungroup | uniquify}
```

Edits a gate-level design.

Options and Arguments

<code>bitblast_all_ports</code>	Bitblasts all ports of a design or subdesign.
<code>bitblast_port</code>	Bitblasts a port_bus (or subport_bus) of a design or hierarchical instance.
<code>connect</code>	Connects a pin, port or subport to another pin, port or subport.
<code>dedicate_subdesign</code>	Replaces a subdesign of instances with a dedicated copy.
<code>delete</code>	Removes an object from the design hierarchy.
<code>disconnect</code>	Disconnects a pin, port or subport.
<code>group</code>	Builds a level of hierarchy around instances.
<code>hier_connect</code>	Connects two objects in different levels of the hierarchy.
<code>new_design</code>	Creates a new design.
<code>new_instance</code>	Creates a new instance.
<code>new_port_bus</code>	Creates a new port_bus on a design.
<code>new_primitive</code>	Creates a new unmapped primitive instance.
<code>new_subport_bus</code>	Creates a new subport_bus on a hierarchical instance.
<code>ungroup</code>	Flattens a level of hierarchy
<code>uniquify</code>	Eliminates sharing of subdesigns between instances

Related Information

Related commands:

[edit_netlist bitblast_all_ports](#) on page 948

[edit_netlist bitblast_port](#) on page 949

Command Reference for Encounter RTL Compiler

Design Manipulation

[edit netlist connect](#) on page 950
[edit netlist dedicate subdesign](#) on page 952
[edit netlist delete](#) on page 953
[edit netlist disconnect](#) on page 954
[edit netlist group](#) on page 956
[edit netlist hier connect](#) on page 958
[edit netlist new design](#) on page 959
[edit netlist new instance](#) on page 960
[edit netlist new port bus](#) on page 962
[edit netlist new primitive](#) on page 963
[edit netlist new subport bus](#) on page 965
[edit netlist uniquify](#) on page 967
[edit netlist group](#) on page 956
[edit netlist ungroup](#) on page 966
[ui respects preserve](#)

Affected by this attribute:

edit_netlist bitblast_all_ports

```
edit_netlist bitblast_all_ports {design|subdesign}...
```

Bitblasts all ports of the specified design or subdesign. This command is available after elaboration. The name of the bitblasted ports will follow the nomenclature specified by the `bit_blasted_port_style` attribute. The default style is:

```
%s_%d
```

Options and Arguments

{*design|subdesign*} Specifies the design or subdesign in which the ports should be bitblasted.

Example

In the following example, the Verilog design `top` has a four-bit input port named `AI`:

```
AI[0:3]
```

The `edit_netlist bitblast_all_ports` command is issued on the design `top`, bitblasting the `AI` port:

```
...
rc:/> synthesize
...
rc:/> edit_netlist bitblast_all_ports top
rc:/> ls /designs/top/ports_in

AI_0 AI_1 AI_2 AI_3
```

Related Information

Affected by this attribute: [bit_blasted_port_style](#)

edit_netlist bitblast_port

```
edit_netlist bitblast_port
  {port_bus | subport_bus}
  {design | instance}
```

Bitblasts the specified port_bus (or subport_bus) of the specified design (or hierarchical instance). This command is available after elaboration. The name of the bitblasted ports will follow the nomenclature specified by the `bit_blasted_port_style` attribute. The default style is:

```
%s %d
```

Options and Arguments

<i>design</i>	Specifies the design to which the port_bus belongs.
<i>instance</i>	Specifies the hierarchical instance to which the subport_bus belongs.
{ <i>port_bus</i> <i>subport_bus</i> }	Specifies the name of the port_bus or subport_bus to be bitblasted.

Related Information

Affected by this attribute: [bit_blasted_port_style](#)

edit_netlist connect

```
edit_netlist connect
  {constant|pin|pgpin|port|subport}
  {constant|pin|pgpin|port|subport}
  [-net_name string]
```

Connects the two specified objects, and anything to which they might already be connected.

You can create nets that have multiple drivers, and you can create combinational loops.

The `logic0` and `logic1` pins are visible in the directory so that you can connect to them and disconnect from them. They are in a directory called `constants` and are called `1` and `0`. The following example shows how the top-level `logic1` pin appears in a design called `add`:

```
/designs/add/constants/1
```

The following example shows how a `logic0` pin appears deeper in the hierarchy:

```
/designs/add/instances_hier/bad/constants/0
```

Each level of hierarchy has its own dedicated logic constants that can only be connected to other objects within that level of hierarchy.

The command has a number of limitations. Violation of the following limitations will generate error messages and cause the command to fail. You cannot connect

- Pins, ports, or subports that are in different levels of hierarchy.
- Pins, ports, or subports that are already connected
- An object to itself.
- An object that is driven by a logic constant to an object that already has a driver. This prevents you from shorting the logic constant nets together.
- Objects if it would require a change to a preserved module.

Options and Arguments

<code>constant</code>	Specifies the name of the constant to connect.
<code>-net_name string</code>	Specifies the user-defined name of the net.
<code>pin</code>	Specifies the name of an instance pin to connect.
<code>pgpin</code>	Specifies the name of the power or ground pin to connect.

Command Reference for Encounter RTL Compiler

Design Manipulation

port

Specifies the name of a design port to connect.

subport

Specifies the name of a subport (port on a hierarchical instance) to connect.

Example

- In the following example, A and B are already connected and C and D are already connected. When you connect A and C, the result is a net connecting A, B, C, and D.

```
rc:/designs/alu/ports_in> edit_netlist connect A C  
/designs/alu/nets/A_
```

Related Information

Related command:

[edit_netlist disconnect on page 954](#)

edit_netlist dedicate_subdesign

```
edit_netlist dedicate_subdesign instance [instance]...
```

Creates a new subdesign by copying the subdesign that is common to the listed hierarchical instances. The command returns the path to the newly created subdesign.

The creation of a new subdesign allows you to make changes that affect a limited set of instances instead of all instances of the original subdesign.

Options and Arguments

<i>instance</i>	Specifies the name of a hierarchical instance for which you want a dedicated subdesign. You must specify a list of instances that share the same subdesign.
-----------------	--

Example

- In the following example the design top contains a module *sub* that has been instantiated five times in the design. The instance names are *sub1*, *sub2*, *sub3*, *sub4*, and *sub5*. To create a separate subdesign for instances *sub1* and *sub2*, enter the following command:

```
rc:/> edit_netlist dedicate_subdesign {/designs/top/instances_hier/sub1 \
    /designs/top/instances_hier/sub2}
```

Note: If you would execute the `edit_netlist dedicate_subdesign` command on the remaining three instances, no new subdesign would be created because they already share a subdesign that is not used by any other instances.

edit_netlist delete

`rm object... [-quiet]`

Removes an object from the design hierarchy. This command is similar to its UNIX counterpart.

For a current list of the objects that can be removed, refer to the command help.

If the hierarchical pin or port bus object has a net connection, the net is disconnected first and then the object is removed.

If you remove a design, the CPF-related information will also be removed from the design hierarchy.

Note: This command does not work on the pin or port object.

Alias for rm.

edit_netlist disconnect

```
edit_netlist disconnect {pin|pgpin|port|subport}
```

Disconnects a single pin, port, or subport from each object it is connected to. For example, if A, B, and C are connected together and you disconnect A, then B and C remain connected to each other, but A is now left unconnected.

You cannot disconnect an object that would require changes to a preserved module.

You cannot disconnect a generic constant (1 or 0) pin of the module but you can disconnect the loads from that pin.

You can disconnect an object that is not currently connected to anything else. In that case nothing happens.

Options and Arguments

<i>pin</i>	Specifies the name of an instance pin to disconnect.
<i>pgpin</i>	Specifies the name of the power or ground pin to disconnect.
<i>port</i>	Specifies the name of a design port to disconnect.
<i>subport</i>	Specifies the name of a subport (port on a hierarchical instance) to disconnect.

Examples

- The following example disconnects input port data [4].

```
rc:/designs/alu/ports_in> edit_netlist disconnect data[4]
```

- The following example shows how you can disconnect a constant pin 1 from all its loads.

```
set cnet [get_attr net /designs/test/constants/1]
set all_loads [get_attr loads $cnet]
foreach_load $all_Loads {
    edit_netlist disconnect $load
}
```

Note: If the constant pin has a large number of loads, disconnecting each of these loads may impact runtime.

Command Reference for Encounter RTL Compiler

Design Manipulation

Related Information

[Using the edit_netlist Command in Design for Test in Encounter RTL Compiler](#)

Related command: [edit_netlist connect](#) on page 950

edit_netlist group

```
edit_netlist group -group_name group_name
    instance [instance]...
```

Creates a level of the design hierarchy by grouping the specified instances. You can only group instances that belong to the same hierarchy.

Options and Arguments

-group_name *group_name*

Specifies the name of the module that groups the specified instances.

By default, the resulting module will have an instance name consisting of the specified group name with the suffix *i*, and is placed in the *instances_hier* directory. The suffix is used to indicate that this hierarchy is the result of a group command.

You can change the suffix with the *group_instance_suffix* attribute.

instance

Specifies the name of an instance to add to the specified group. You need to specify at least one instance.

Examples

- The following example groups instances *accum_1* and *averg_1* into one module *my_module*.

```
rc:/> edit_netlist group -group_name my_module accum_1 averg_1
/designs/alu/instances_hier/my_modulei
```

- The following command returns an error because the specified instances do not belong to the same hierarchy.

```
rc:/> edit_netlist group [find / -instance m5] [find / -instance m3_0]
Error   : Not all instances belong to the same hierarchy. [TUI-234] [edit_netlist
group]
      : Instance '/designs/m1/instances_hier/m2/instances_hier/m3/
instances_hier/m4/instances_hier/m5' is part of (sub)design 'm4'. Instance '/
designs/m1/instances_hier/m2/instances_hier/m3/instances_hier/m3_0' is not part of
(sub)design 'm4'.
      : The 'edit_netlist group' command can only group instances contained
within the same hierarchy.
```

Command Reference for Encounter RTL Compiler

Design Manipulation

Related Information

[Grouping and Ungrouping Objects](#) in *Using Encounter RTL Compiler*

Related command: [edit_netlist ungroup](#) on page 966

Affected by these attributes: [group_generate_portname_from_netname](#)
[group_instance_suffix](#)

edit_netlist hier_connect

```
edit_netlist hier_connect
  {constant|subport|port|pin|pgpin}
  {constant|subport|port|pin|pgpin}
  [-prefix string]
  [-in_prefix string] [out_prefix string]
```

Connects two objects in different levels of the hierarchy.

Note: You should specify the driving pin before the load pin.

Options and Arguments

{constant|subport|port|pin|pgpin}

Specifies the object to connect.

-in_prefix string Specifies the prefix for new input ports.

-out_prefix string Specifies the prefix for new output ports.

-prefix string Specifies the prefix for ports and nets.

Example

The following command connects the input pins of two inverters in the hierarchical instances aa1 and aa2.

```
edit_netlist hier_connect [find . -pin aa1/inv/in_0] [find . -pin aa2/inv/in_0]
```

Related Information

[Using the edit_netlist Command in Design for Test in Encounter RTL Compiler](#)

Related command: [edit_netlist connect](#) on page 950

edit_netlist new_design

```
edit_netlist new_design -name string [-quiet]
```

Creates a new design at the same level as the existing top-level design.

The new design is created in the /designs directory. Once the design is created, you can specify instances, port_bus, and so on.

Options and Arguments

<code>-name <i>string</i></code>	Specifies the name of the new design.
<code>-quiet</code>	Suppresses the warning messages regarding naming conflicts.

Examples

- The following example creates a new design called DESIGNA.

```
rc:/> edit_netlist new_design -name DESIGNA
```

The new design DESIGNA, will be created in the /designs directory. Once the design is created, the instances, port_bus, and so on can be specified.

- The following example tries to create a new design called TEST. However, a design by that name already exists. In such cases, a number will be appended to the end of the specified name and an error message indicating the naming conflict will be printed. The following example specifies the -quiet option to suppress this warning.

```
rc:/> edit_netlist new_design -name TEST -quiet  
/designs/TEST1
```

The name given to the new design in this case is TEST1.

Related Information

Related command:

[edit_netlist new_port_bus on page 962](#)

edit_netlist new_instance

```
edit_netlist new_instance [-name string]
  {design|subdesign|libcell}
  {subdesign|design} [-quiet]
```

Creates a specified instance type in a specified level of design hierarchy. You can instantiate inside a top-level design or a subdesign.

- You cannot instantiate objects that require a change to a preserved module.
 - You cannot create a hierarchical loop.
- If subdesign A contains subdesign B, you cannot instantiate A underneath B.

Options and Arguments

<i>-name string</i>	Specifies the name of the new instance.
{ <i>design subdesign libcell</i> }	Specifies the object to instantiate.
<i>-quiet</i>	Suppresses the warning messages regarding naming conflicts.
{ <i>subdesign design</i> }	Specifies the name of the design or subdesign in which you want to instantiate the object.

Examples

- The following example creates a new instance called TEST_SUB under the TEST design:

```
rc:/> edit_netlist new_instance -name TEST_SUB /designs/TEST \
      /designs/TEST
rc:/> ls /designs/TEST/instances_hier/
      /designs/TEST/instances_hier/TEST_SUB
```

- The following example tries to create a new subdesign called TEST_SUB under the TEST design. However, a subdesign by that name already exists. In such cases, a number will be appended to the end of the specified name and an error message indicating the naming conflict will be printed. The following example specifies the -quiet option to suppress this warning.

```
rc:/> edit_netlist new_instance -name TEST_SUB /designs/TEST -quiet \
      /designs/TEST
      /designs/TEST/instances_hier/TEST_SUB3
```

The name given to the new subdesign in this case is TEST_SUB3.

Command Reference for Encounter RTL Compiler

Design Manipulation

Related Information

Related command: [edit_netlist new_subport_bus](#) on page 965

edit_netlist new_port_bus

```
edit_netlist new_port_bus -name string
    [-left_bit integer] [-right_bit integer]
    {-input|-output|-input -output}
    [design]
```

Creates a `port_bus` object in a design. The command can also create a single port by omitting both the `-left_bit` and `-right_bit` options.

Options and Arguments

<i>design</i>	Specifies the name of the design for which to create the <code>port_bus</code> . The design name can be omitted if there is only one top-level design.
<code>-input</code>	Creates an input <code>port_bus</code> .
<code>-input -output</code>	Creates a bidirectional <code>port_bus</code> .
<code>-left_bit <i>integer</i></code>	Specifies the leftmost bit index of the bus.
<code>-name <i>string</i></code>	Specifies the name of the new <code>port_bus</code> .
<code>-output</code>	Creates an output <code>port_bus</code> .
<code>-right_bit <i>integer</i></code>	Specifies the rightmost bit index of the bus.

Example

- The following example creates a single input port named `a_in`:

```
rc:/> edit_netlist new_port_bus -name a_in -input
```

Related Information

Related command: [edit_netlist new_design](#) on page 959

edit_netlist new_primitive

```
edit_netlist new_primitive [-name string] [-inputs integer]  
[-quiet] logic_function {design|subdesign}
```

Creates an unmapped primitive cell in a design or a subdesign.

Options and Arguments

{*design|subdesign*}

Specifies the name of the design or subdesign in which you want to instantiate the primitive cell.

-inputs *integer* Specifies the number of input pins to create for the primitive cell.

logic_function Specifies the logic function of the primitive cell. You can specify any of the following:

and	latch	notif1
buf	nand	or
bufif0	nor	xnor
bufif1	not	xor
d_flop	notif0	

-name *string* Specifies the name of the new primitive cell.

-quiet Suppresses the warning messages regarding naming conflicts.

Examples

- The following example creates a new buffer instance called I101 in design DESIGNA:

```
rc:/> edit_netlist new_primitive -name I101 buf DESIGNA
```

The new instance, I101, is created in the /designs/DESIGNA/instances_comb directory.

A sequential primitive (d_flop or latch) will be created in the instances_seq directory.

Command Reference for Encounter RTL Compiler

Design Manipulation

- The following example tries to create a new buffer instance called I101. However, a buffer by that name already exists. In such cases, a similar name will be chosen and an error message indicating the naming conflict will be printed. The following example specifies the -quiet option to suppress this warning:

```
rc:/> edit_netlist new_primitive -name I101 buff TEST -quiet  
/designs/TEST/instances_comb/I1
```

The name given to the new buffer in this case is I1.

edit_netlist new_subport_bus

```
edit_netlist new_subport_bus -name string
    [-left_bit integer] [-right_bit integer]
    {-input|-output|-input -output}
instance
```

Creates a `subport_bus` in a design. The command can also create a single subport by omitting both the `-left_bit` and `-right_bit` options.

Options and Arguments

<i>design</i>	Specifies the name of the instance for which to create the <code>subport_bus</code> .
<code>-input</code>	Creates an input <code>subport_bus</code> .
<code>-input -output</code>	Creates a bidirectional <code>subport_bus</code> .
<code>-left_bit <i>integer</i></code>	Specifies the leftmost bit index of the <code>subport_bus</code> .
<code>-name <i>string</i></code>	Specifies the name of the new <code>subport_bus</code> .
<code>-output</code>	Creates an output <code>subport_bus</code> .
<code>-right_bit <i>integer</i></code>	Specifies the rightmost bit index of the <code>subport_bus</code> .

Example

- The following example creates a single input subport named `a_in`:

```
rc:/> edit_netlist new_subport_bus -name a_in -input
```

Related Information

Related command: [edit_netlist new_instance](#) on page 960

edit_netlist ungroup

```
edit_netlist ungroup [-prefix string] instance...
```

Removes a level of the design hierarchy.

Large numbers of small hierarchical blocks in a design can sometimes limit optimization since the hierarchical boundaries must be preserved. Many hierarchical blocks may also increase the memory usage since information about each block and port names must be stored. The ungroup command provides a mechanism to remove these unwanted levels of hierarchy.

Options and Arguments

<i>instance</i>	Specifies the hierarchical instance for which to remove one level of the hierarchy. The components of the specified instance then become instances in the parent block.
-prefix	Specifies a prefix for the ungrouped instances.

Examples

- The following example ungroups all hierarchical instances whose names end in _little:

```
rc:/> edit_netlist ungroup [find / -instance *_little]
```
- The following example ungroups the instance inst1 and specifies that the resulting ungrouped instances of inst1 have a prefix of inst1_test. Using the prefix allows you to identify from which instance the ungrouped instances originated:

```
rc:/designs/test/instances_hier> edit_netlist ungroup -prefix inst1_test \
inst1
rc:/designs/test/instances_comb> ls
```

```
inst1_test_g1/  inst1_test_g2/  inst1_test_g3/
```

Related Informations

[Grouping and Ungrouping Objects in Using Encounter RTL Compiler](#)

Related command: [edit_netlist group](#) on page 956

edit_netlist uniquify

```
edit_netlist uniquify  
  {subdesign|design} [-verbose]
```

Uniquifies the instances under the specified design or subdesign. Uniquification is the process of creating a new subdesign for an instance or a group of instances. The newly created subdesign is merely a copy of the subdesign to which the original instance or group of instances were associated. That is, an instance or a group of instances will now be a part of their own, *unique* subdesign. The newly created subdesign will usually maintain the original design or subdesign name followed by a number suffix.

Note: Preserved modules cannot be uniquified.

Options and Arguments

{*design|subdesign*}

The instances under the specified design or subdesigns will be uniquified.

-verbose

Prints out the uniquified instances and their corresponding subdesign names.

Examples

- The following example has a top level design called `top` with two subdesigns named `A` and `B`:

```
rc:/designs/top/subdesigns> ls  
./      A/      B/
```

In order to uniquify the subdesigns, issue the `edit_netlist uniquify` command on `top`:

```
rc:/designs/top/subdesigns> edit_netlist uniquify /designs/top  
rc:/designs/top/subdesigns> ls  
./      A/      A_1/      B/      B_1/
```

- The following example shows how to uniquify all subdesigns except for subdesign `mysubdesign`.

```
set_attribute preserve true [find / -subdesign mysubdesign]  
foreach el [find / -subdesign *] {  
    edit_netlist uniquify $el  
}
```

Command Reference for Encounter RTL Compiler

Design Manipulation

group

```
edit_netlist group -group_name group_name  
           instance [instance]...
```

Creates a level of the design hierarchy by grouping the specified instances. You can only group instances that belong to the same hierarchy.

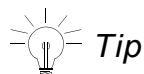
Alias for [edit_netlist group](#).

insert_tiehilo_cells

```
insert_tiehilo_cells
  [-hilo libcell] [-hi libcell] [-lo libcell]
  [-allow_inversion] [-maxfanout integer]
  [-all] [-skip_unused_hier_pins] [-place_cells]
  [-verbose] [subdesign | design]
```

Ties the constants 1'b0 and 1'b1 in the netlist to tie high and tie low cells, respectively. In multiple supply voltage (MSV) designs, this command inserts cells by domain. It skips scan pins, preserved pins, preserved nets, and modules by default. Scan pins can be connected by using the -all option.

Use the ui_respects_preserve root attribute to override preserve settings.



Tip
Ensure that the specified tie high and tie low cells are usable. Make sure that both the preserve and avoid libcell attributes are set to false on the specified cells.

Options and Arguments

-all	Inserts tie hi/lo cells without skipping scan pins.
-allow_inversion	Allows to use a tie cell with inverter if either the tie high or tie low cell cannot be found in the library.
-hi libcell	Specifies a cell for constant 1s. By default, the first appropriate cell will be used.
-hilo libcell	Specifies a high-low cell to connect constants. By default, the first appropriate cell will be used.
-lo libcell	Specifies a cell for constant 0s. By default, the first appropriate cell will be used.
-maxfanout integer	Specify the maximum fanout allowed per tie cell. If this option is not specified, there is no constraint on the fanout.
-place_cells	Places the inserted tie cells.
-skip_unused_hier_pins	Skips hierarchical constant connected pins which are not used inside the module.

Command Reference for Encounter RTL Compiler

Design Manipulation

{subdesign | design}

Specifies the design or subdesign in which to insert constants.

If you omit the design name, the top-level design of the current directory of the design hierarchy is used.

-verbose

Provides detailed information of the preserved and scan pins that were skipped in the tie hi/lo cell insertion process.

Examples

- The following example ties the constant 1s and 0s to the cells named TIEHI and TIELOW, respectively. The maximum fanout per tie cell is 10. Using the **-verbose** option shows that two scan pins were skipped:

```
rc:/> insert_tiehilo_cells -hi TIEHI -lo TIELO -maxfanout 10 -verbose
pin: /libraries/slow/libcells/TIELO/Y function: 0
pin: /libraries/slow/libcells/TIEHI/Y function: 1
Connecting all 1'b0 and 1'b1 to TIELO/TIEHI cells.
tielo cell is /libraries/slow/libcells/TIELO , tiehi_cell is /libraries/slow/
libcells/TIEHI

Info : 2 scan pins which are loads of '0' in /designs/m1 are skipped.
Use the '-all' option to avoid skipping of scan pins.
/designs/m1/instances_seq/foo/bx_reg/pins_in/SE
/designs/m1/instances_seq/tm_reg/pins_in/SE

Done connecting 1'b0 and 1'b1 to TIELO/TIEHI cells
```

- The following example shows two modules, **U1** and **top**. When the **insert_tiehilo_cells -skip_unused_hier_pins** command is used, the pin **B** of the instantiation of **U1** in module **top** will be skipped.

```
module U1(A, B, C, Sel, Z);
    input A, B, C;
    input [1:0] Sel;
    output Z;
    wire A, B, C;
    wire [1:0] Sel;
    wire Z;
    wire n_0, n_1, n_2, n_3, n_4, n_5;
    NAND2X1 g27(.A (n_5), .B (n_4), .Y (Z));
    NAND2X1 g28(.A (n_3), .B (n_2), .Y (n_5));
    NAND2X1 g29(.A (n_1), .B (A), .Y (n_4));
    NOR2X1 g30(.A (n_0), .B (Sel[0]), .Y (n_3));
    INVX1 g31(.A (n_1), .Y (n_2));
    NOR2X1 g32(.A (Sel[1]), .B (Sel[0]), .Y (n_1));
    INVX1 g33(.A (C), .Y (n_0));
endmodule

module top(a, b, c, sel, z);
    input a, b, c;
    input [1:0] sel;
    output z;
```

Command Reference for Encounter RTL Compiler

Design Manipulation

```
wire a, b, c;
wire [1:0] sel;
wire z;
U1 inst_U1(.A (a), .B (1'b0), .C (c), .Sel (sel), .Z (z));
endmodule
```

Related Information

[Removing Assign Statements](#) in *Using Encounter RTL Compiler*

Related attributes:

[ui_respects_preserve](#)
[use_tiehilo_for_const](#)

Command Reference for Encounter RTL Compiler

Design Manipulation

mv

```
mv object new_name [-flexible] [-slash_ok]
```

Renames an object in the design hierarchy. This command is similar to its UNIX counterpart.

You can rename the following objects:

- design
- instance
- isolation_rule
- level_shifter_group
- level_shifter_rule
- library_domain
- net
- port_bus
- power_domain
- scan_chain
- scan_segment
- subdesign
- subport_bus
- test_clock
- test_clock_domain
- test_signal

Options and Arguments

<code>-flexible</code>	Indicates to be flexible for renaming when there is a collision.
<code>new_name</code>	Specifies the new name for the specified object.
<code>object</code>	Specifies the object to rename.
<code>-slash_ok</code>	Indicates that the destination name can have embedded slashes.

Examples

- The following example changes the name of design `comp` to `comp_test`.

```
rc:/designs> ls  
comp  
rc:/designs> mv comp comp_test  
rc:/designs> ls  
comp_test
```

- The following example changes the subdesign `mux` to `muxYYY`. You do not have to specify the path name of the target object, just the basename:

```
rc:/> mv /designs/dpldalgns/subdesigns/mux muxYYY  
rc:/> ls /designs/dpldalgns/subdesigns/  
muxYYY
```

- The following example attempts to rename instance `aluout_reg_0` to an already existing instance `aluout_reg_1`. Using the `-flexible` option, the instance gets renamed to `aluout_reg585`, which does not cause a conflict:

```
rc:/designs/alu/instances_seq> mv aluout_reg_0 aluout_reg_1 -flexible  
/designs/alu/instances_seq/aluout_reg585  
rc:/designs/alu/instances_seq> ls  
. / aluout_reg_1/ aluout_reg_3/ aluout_reg_5/ aluout_reg_7/  
aluout_reg585/ aluout_reg_2/ aluout_reg_4/ aluout_reg_6/ zero_reg/
```

Related Information

Related command: [change_names](#) on page 937

Command Reference for Encounter RTL Compiler

Design Manipulation

remove_cdn_loop_breaker

```
remove_cdn_loop_breaker  
    -instances instance list design
```

Removes the specified loop breaker buffers added by the timing engine and restores the loop.

Options and Arguments

design Specifies the design for which the loop breakers must be removed.

-instances *instance list*

Specifies the loop breakers instances that you want to remove.

If this option is not specified, all instance loop breakers will be removed.

Example

The following example first shows the loop breakers inserted, then removes the loop breakers and lists the report again.

```
rc:/> report cdn_loop_breaker
=====
Generated by:           version
Generated on:          date
Module:                loop
Technology library:    tutorial 1.1
Operating conditions: typical_case (balanced_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====

CDN Loop breaker      Driver     Load
-----
inst1/cdn_loop_breaker inst1/i1/Y i0/B

rc:/> remove_cdn_loop_breaker -instance [find / -inst inst1/cdn_loop_breaker]
rc:/> report cdn_loop_breaker
=====
Generated by:           Encounter(R) RTL Compiler 8.1.200
Generated on:          Oct 20 2008 03:53:12 PM
Module:                loop
Technology library:    tutorial 1.1
Operating conditions: typical_case (balanced_tree)
Wireload mode:         enclosed
Area mode:             timing library
=====

No loop breakers to report
```

Command Reference for Encounter RTL Compiler

Design Manipulation

Related Information

Related command: [report cdn_loop_breaker](#) on page 389

Command Reference for Encounter RTL Compiler

Design Manipulation

reset_design

```
reset_design [-verbose] [design]
```

Removes all the user specified timing and DFT objects and returns all attributes to their default values for the specified design. Alternatively, this command removes every clock domain, cost group, exception, external delay, scan chain, scan segment, test clock domain, and test signal while returning all attributes on the design to their default values.

Options and Arguments

design Specifies the particular design to reset when there are multiple designs.

-verbose Prints messages indicating that the command was successful.

Example

- The following examples illustrates that the `reset_design` command has eliminated all external delays in the design:

```
rc:/designs/phoenix/timing/external_delays> ls  
in 1  out 2
```

```
rc:/designs/phoenix/timing/external_delays> reset_design phoenix
rc:/designs/phoenix/timing/external_delays> ls
./
```

rm

```
rm object... [-quiet]
```

Removes an object from the design hierarchy. This command is similar to its UNIX counterpart.

For a current list of the objects that can be removed, refer to the command help.

If the hierarchical pin or port bus object has a net connection, the net is disconnected first and then the object is removed.

If you remove a design, the CPF-related information will also be removed from the design hierarchy.

Note: The `rm` command does not work on the pin or port object.

Options and Arguments

<i>object</i>	Specifies the object that you want to remove. Check the command help for a list of the removable object types.
-quiet	Suppresses those messages that indicate which objects are being removed. Alternatively, when removing an object, an information message will not be printed.

Examples

- The following example finds the clock objects in the design:

```
rc:/> find . -clock *
/designs/comp/timing/clock_domains/domain_1/clock1
```

- The following example uses the result of the `find` command to remove the clock. This command also removes all dependent objects. A subsequent `find` cannot find any clock objects.

```
rc:/> rm [find . -clock *]
Info    : Removing a clock object [TIM-102]
        : The clock name is 'clock'
        Removing external delay 'in_del_1'.
        Removing external delay 'ou_del_1'.
rc:/> find . -clock *
I cannot find any clock named * here
Failed on find . -clock *
```

ungroup

```
ungroup
[ -all | -flatten instance...
| -threshold integer | instance...]
[-simple] [-only_user_hierarchy]
[-prefix string]
[-exclude instances]
```

Ungroups the specified instances. Ungrouping dissolves the hierarchy and moves the contents of a subdesign into its parent directory. By default, an instance is named by concatenating its name to its parent's name.

Options and Arguments

-all	Ungroups all instances at the current level.
-exclude <i>instances</i>	Specifies a list of instances that should not be ungrouped.
-flatten	Recursively ungroups all the specified instances.
<i>instance</i>	Specifies the instance or instances to be ungrouped.
-only_user_hierarchy	Ungroups only those hierarchies created by the user. The hierarchies created by the tool are preserved.
-prefix <i>string</i>	Adds the specified prefix to the names of the wires, nets, and instances created as a result of flattening the hierarchical instance(s).
-simple	Prevents the use of a complex new instance name during ungrouping. This option has the same effect as the <code>edit_netlist ungroup</code> command.
-threshold <i>integer</i>	Ungroups only those hierarchical instances that have a cell count equal to or less than the specified integer.

Examples

- The following example ungroups the instance named CRITICAL_GROUP:

```
rc:/> ungroup [find / -instance CRITICAL_GROUP]
```

Command Reference for Encounter RTL Compiler

Design Manipulation

- In the following example, every instance under the `inst` hierarchical instance will be ungrouped except the `inst_sub2` instance:

```
rc:/designs/ksable_hier/instances_hier/inst/instances_hier> ls  
./      inst_sub1/      inst_sub2/      inst_sub3  
rc:/designs/ksable_hier/instances_hier/> ungroup inst -flatten -exclude \  
[find . -instance inst/inst_sub2]
```

Related Information

Related command: [edit_netlist ungroup](#) on page 966

uniquify

```
edit_netlist uniquify  
  {subdesign|design} [-verbose]
```

Uniquifies the instances under the specified design or subdesign. Uniquification is the process of creating a new subdesign for an instance or a group of instances. The newly created subdesign is merely a copy of the subdesign to which the original instance or group of instances were associated. That is, an instance or a group of instances will now be a part of their own, *unique* subdesign. The newly created subdesign will usually maintain the original design or subdesign name followed by a number suffix.

Note: Preserved modules cannot be uniquified.

Alias for [edit_netlist uniquify](#).

Customization

- [add_command_help](#) on page 982
- [define_attribute](#) on page 983
- [mesg_make](#) on page 987
- [mesg_send](#) on page 989
- [parse_options](#) on page 990

add_command_help

add_command_help command_name help category

Adds a short help message for a new command to RTL Compiler's online help system. Examples of *add_command_help* can be found in the installation *lib/cdn/rc* directory.

Options and Arguments

<i>category</i>	Specifies the category to which this command should be added. You can specify an existing category or a new one. To see a list of all existing categories, type <i>help</i> .
<i>command_name</i>	Specifies the name of the command. It must be a Tcl procedure that you defined previously.
<i>help</i>	Lists the help text to be displayed when the <i>help</i> command is used. Use a string.

Examples

- The following example adds help for the *hello_world* command:

```
rc:/> proc hello_world {hello_world} { echo "Hello world" }
rc:/> add_command_help "hello_world" "Says hello to the whole world"
my_category
rc:/> help hello_world
That command is:
```

```
my_category
=====
hello_world Says hello to the whole world
Command details:
Hello world
```

define_attribute

```
define_attribute string
    -category string -data_type string -obj_type string
    [-hidden] [-help_string string]
    [-check_function string] [-compute_function string]
    [-set_function string] [-default_value string]
    [-skip_in_db]
```

Creates a new, user-defined attribute with the specified characteristics. These attributes will also be written out if you use the `write_script` command.

Options and Arguments

`-category string`

Defines the category of the attribute. Categories group attributes that perform similar functions whereas object types describe where in the design an attribute is valid. You can specify any category name: both new and existing category names are valid.

`-check_function string`

Specifies a previously defined Tcl procedure's name in order to ensure that the newly defined attribute is valid. The Tcl procedure should be of the form:

```
proc {object value}
```

The Tcl procedure returns 1 for a valid value, and 0 for an invalid value.

`-compute_function string`

Specifies a previously defined Tcl procedure's name in order to get the newly defined attribute's value later (with the `get_attribute`) command. The Tcl procedure should be of the form:

```
proc {object}
```

When you use this option, the attribute becomes a read-only attribute because its value is always computed.

Command Reference for Encounter RTL Compiler

Customization

-data_type *string*

Defines the data type of the attribute. Possible data types are:

- boolean
- fixed point
- double
- integer
- string

-default_value *string*

Specifies a default value for the attribute.

-help_string *string*

Specifies the help text for the attribute.

-hidden Specifies whether the defined attribute is a hidden attribute.

-more_help_string *string*

Specifies an extended help string.

-obj_type *string*

Specifies the object type of the attribute. All valid object types can be found by typing `find -help` at the RTL Compiler prompt.

-obsolete Specifies whether the defined attribute is obsolete.

-set_function Specifies a previously defined Tcl procedure's name. This option allows you to override user-defined values provided it conforms to the parameters in the Tcl procedure you created. The Tcl procedure should be of the form:

```
proc {object new_value current_value}
```

-skip_in_serialization

Prevents the `write_db` command to write out the defined attribute.

string

Specifies the name of the attribute.

Examples

- The following example defines a boolean attribute named `new_libpin` for a new category named `my_libpin`:

```
rc:/> define_attribute -data_type boolean -obj_type libpin -category my_libpin
new_libpin
rc:/> get_attribute new_libpin * -help
...
attribute category: my_libpin
attribute name: new_libpin
    category: my_libpin ()
object type: libpin
access type: read-write
    data type: boolean
default value:
    help:
```

- The following example creates a Tcl procedure, `check_fxn`, and then creates an attribute named `test_check`. The `-check_function` option is specified so that the `test_check` attribute can be tested for validity when it is specified later.

```
rc:/> proc check_fxn {obj val} {
==>     if {$val < 0} {
==>         return 0
==>     }
==>     return 1
==> }

rc:/> define_attribute test_check -obj_type root -data_type integer \
    -category test -help_string "test check function" \
    -check_function check_fxn
/object_types/root/attributes/test_check
```

The following command is a valid use of the newly created `test_check` attribute:

```
rc:/> set_attribute test_check 1 /
Setting attribute of root '/': 'test_check' = 1
```

The following command would be an invalid use:

```
rc:/> set_attribute test_check -1 /
Error : The data value for this attribute is invalid. [TUI-24] [set_attribute]
       : The value '-1' cannot be set for attribute 'test_check'.
       : To see the usage/description for this attribute, type 'set_attribute
-h <attr_name> *'.
```

- The following example creates a Tcl procedure, `set_fxn`, and then creates an attribute named `test_set`. The `-set_function` option is specified so that you can change the value of the `test_set` attribute (provided it is valid):

```
rc:/> proc set_fxn {obj new_val cur_val} {
==>     if {$new_val > $cur_val} {
==>         return $new_val
==>     }
==>     return $cur_val
==> }
```

Command Reference for Encounter RTL Compiler

Customization

```
rc:/> define_attribute test_set -obj_type root -data_type integer \
         -category test -help_string "test set function" -set_function set_fxn
/object_types/root/attributes/test_set
```

The `test_set` attribute will be changed to 1. It is valid, since there was no previous value:

```
rc:/> set_attribute test_set 1
Setting attribute of root '/': 'test_set' = 1
```

The following command changes the attribute value to 2. Again, this is valid because it falls within the definition of the previously defined Tcl procedure, `set_fxn`:

```
rc:/> set_attribute test_set 2 /
Setting attribute of root '/': 'test_set' = 2
```

The attribute's value will not be changed in the following example. The value will remain at 2:

```
rc:/> set_attribute test_set 0 /
Setting attribute of root '/': 'test_set' = 2
```

- The following example creates a Tcl procedure, `compute_fxn`, which always returns the value of 42. The `define_attribute` command then creates an attribute named `test_compute`. The `-compute_function` option is specified so that you can obtain the `test_compute` attribute's value later:

```
rc:/> proc compute_fxn {obj} {
==>   return 42
==> }
rc:/> define_attribute test_compute -obj_type root -data_type integer \
         -category test -help_string "test compute function" \
         -compute_function compute_fxn
/object_types/root/attributes/test_compute
```

The `test_compute` value will always be 42, as defined in the Tcl procedure:

```
rc:/> get_attribute test_compute /
42

rc:/> set_attribute test_compute 23 /
Error  : The attribute is read-only. [TUI-26] [set_attribute]
        : attribute: 'test_compute', object type: 'root'
        : Cannot set or reset read-only attributes.
Failed on set_attribute test_compute 23
```

mesg_make

```
mesg_make -group string [-internal_group] -id number
    -short_desc string -long_desc string
    {-error|-warning|-info_priority number}
```

Creates a custom message that can subsequently be accessed with the `mesg_send` command.

Options and Arguments

<code>-error</code>	Creates an error message.
<code>-group <i>string</i></code>	Specifies the group of messages that the new message belongs to. A group groups messages that apply to a certain engine of the tool. For example, the MAP group groups messages issued by the mapper. Note: If you want to create a message for an internal group, you must specify an existing group name.
<code>-id <i>integer</i></code>	Specifies an identification number for the message. The number must be unique for the specified group. If the specified number already exists, you will overwrite the existing message. <i>Default:</i> 1
<code>-info_priority <i>integer</i></code>	Creates an info message with the specified priority. You can specify a number between 2 and 8.
<code>-internal_group</code>	Specifies whether the message belongs to an internal group.
<code>-long_desc <i>string</i></code>	Defines the help of the message.
<code>-short_desc <i>string</i></code>	Specifies the title or the description of the message.
<code>-warning</code>	Creates a warning message.

Example

- The following example creates a message in the test group and assigns it a unique identification number (501) within that group:

```
rc:/> mesg_make -group test -id 501 -short_desc note_bene \
==> -long_desc "search for lost time" -warning
/messages7/test/test-501
rc:/> man test-501
Entry      : test-501
Severity   : Warning
Verbosity   : Message is visible at any 'information_level' above '1'.
Description : note_bene
Help       : search for lost time
```

Related Information

Affects this command: [mesg_send](#) on page 989

mesg_send

`mesg_send message string`

Accesses the various native messages of RTL Compiler and the messages that were created with the `mesg_make` command.

Options and Arguments

<code>-caller string</code>	Specifies the name of the calling procedure.
<code>-file_info string</code>	Specifies to which file the message applies.
<code>message</code>	Identifies the message to be sent. The identification is in the form of <i>group-id</i> . Refer to <code>mesg_make</code> for an explanation of <i>group</i> and <i>id</i>
<code>-newline</code>	Prints a new (empty) line before the message
<code>-object_info string</code>	Prints the object type and
<code>string</code>	Describes the context-specific help.

Examples

- The following example accesses the message created in [Example](#) on page 988 for `mesg_make`:

```
rc:/messages/test> mesg_send test-501 "reminders" -newline
Warning : note_bene [test-501]
          : reminders
          : search for lost time
```

- The following example sends a message after elaboration:

```
rc:/> mesg_send /messages/VHDLPT/VHDLPT-500 "file not found" -caller read_hdl
Error   : Cannot open file. [VHDLPT-500] [read_hdl]
          : file not found
```

Related Information

Affected by this command: [mesg_make](#) on page 987

parse_options

```
parse_options cmd file_var [args] [code var]...
```

Interfaces to the RTL Compiler internal command option parser. The RTL Compiler argument parser provides the following features:

- Checking the correctness for arguments and types. Appropriate messages are issued when the input is incorrect.
- No specific order of arguments is required. For example, `ls -long -attribute` behaves just like `ls -attribute -long`.
- Unique abbreviations of arguments is supported. For example, `ls -l` behaves just like `ls -long`.
- Online help is provided if `-help` is specified.
- Optional file redirection is supported. For example, `report gates >> design.rpt` causes output to be appended to the file `design.rpt`.
- RTL Compiler objects can be implicitly searched for based on their type. For example, `fanout SUB/A[0]` performs an implicit find on the string `SUB/A[0]` and locates the object `/designs/TOP/instances_hier/SUB/pins_in/A[0]`.

You can use the command option parser to make a Tcl procedure behave just like a built-in RTL Compiler command by providing on-line help, finding objects automatically, checking for required options, handling unique argument abbreviations, and handling file redirection.

This command can return one of the following values:

- -2: You asked for help and help was provided. The command returns normally.
- 0: Your options were invalid and the command aborts.
- 1: Your options were valid and the command continues normally.

Options and Arguments

args

Specifies a list containing the options that the user sends to your command. Usually your procedure will be defined like this:

```
proc your_procedure {args} {  
    ... (code that implements the procedure)  
}
```

Command Reference for Encounter RTL Compiler

Customization

You would then pass \$args as the args parameter to parse_options within your procedure.

cmd

Specifies the name of the command whose options to parse.

This name appears in the help listing if a user calls your procedure with -h or if a user does not provide valid arguments.

code

Specifies a string that describes the command argument: flag name, whether it is required or optional, what type of data it accepts, and a short help message about it. The string must be in the form:

```
"(-<name>) ?<x><y><z>(<dirtytypes>) ? <help>"
```

The question marks in the above string mean that these fields of the string are optional.

You cannot specify multiple string values if you are specifying a flag name. That is, if you are specifying a flag, you cannot also specify the som (string, optional, multiple values) or srm (string, required, multiple values) combinations.

<name> is the name of the flag. For example, -number.

<x> is a single character indicating the type of the option:

b	Boolean
d	directory object
f	float
n	integer number
s	string

<y> is a single character indicating whether the option is optional or required

o	optional
r	required
x	obsolete

<z> is a single character indicating whether lists are accepted

m	Accepts multiple values: lists OK
s	Accepts single value only: no lists

Command Reference for Encounter RTL Compiler

Customization

<dirtypes> is a string indicating the types of directory objects the option accepts. This string is required for all arguments that have the *<x>* field set to d and cannot be specified otherwise.

The list of specified directory types can only be separated by vertical bars (|), that is, no spaces allowed, and must be enclosed in parentheses.

<help> is a string that indicates to the user of your command what the purpose of the option is.

file_var

Specifies the name of a variable that holds the file handle if the user calls your procedure with *> file* or *>> file*. Your procedure should always check to see if this variable has been set to something other than 'stdout'. If it has, then your command should send its output to that file handle instead of 'stdout' and you should close the file handle once your command is complete.

var

Specifies the name of the variable that will be set with the parsed result for that argument.

Examples

- The following example shows how file indirection works:

```
rc:/> parse_options hello_world file_var {> ./tmp}
1
rc:/> puts $file_var "Hello world!"
rc:/> close $file_var
```

- The following example shows a required argument with the -design flag that must be the name of a subdesign (block).

```
rc:/> parse_options hello_world file_var {-design addinc65} \
==> "-design drs(subdesign) A module" my_design
1
rc:/> puts $my_design
/designs/alu/subdesigns/addinc65
```

- The following example shows a Boolean argument, a numeric argument, and an un-flagged object argument that can be either a subdesign (block) or an instance. It also shows how the parser accepts abbreviations since the argument passed in is only -t, it still gets recognized as -top.

```
rc:/> set level 300
300
rc:/> parse_options hello_world file_var {-t addinc65} \
==> "-top bos Do the top-level thing" top \
```

Command Reference for Encounter RTL Compiler Customization

```
==> "-level nos The level to work on"    level \
==> "dos(subdesign|instance) An object to work on"    object
1
rc:/> puts $object
/designs/alu/subdesigns/addinc65
rc:/> puts $top
1
rc:/> puts $level
300
```

- The following example shows how online help works:

```
rc:/> parse_options hello_world file_var {-h} \
==> "-top bos Do the top level thing"    top \
==> "-level nos The level to work on"    level \
==> "dos(subdesign|instance) An object to work on"    object
Usage: hello_world [-h]
      -h: this message
hello_world [-top] [-level number] [instance|subdesign] [> file]
      -top: (Boolean) Do the top level thing
      -level:      (integer) The level to work on
                  (instance|subdesign) An object to work on
-2
```

Command Reference for Encounter RTL Compiler

Customization

A

Applets

- [Introduction](#) on page 996
- [applet](#) on page 997
- [applet avail](#) on page 998
- [applet install](#) on page 1000
- [applet list](#) on page 1001
- [applet load](#) on page 1002
- [applet update](#) on page 1003
- [applet version](#) on page 1004
- [applet whatis](#) on page 1005

Introduction

Applets are **non-productized** scripts that can be used for a variety of purposes, such as report generation, template management, text histogram generation, testcase creation, and many others.

The applet infrastructure enables effective management of such infrastructure whether user-generated or Cadence-generated.

All applet commands generate a disclaimer banner, clearly indicating the limitation and conditions of their use.

Command Reference for Encounter RTL Compiler

Applets

applet

```
applet {avail | install | list | load  
| update | version | whatis}
```

Manage applets.

Options and Arguments

avail	Lists all applets: those you have installed as well as those that are available on the server. The listing also includes the version information.
install	Installs the applet tree in the specified location.
list	Lists all applets already loaded.
load	Loads an applet and all of its dependencies.
update	Updates those applets for which a newer version exists on the applet server.
version	Returns the version information for the applet command.
whatis	Displays the description of a specific applet.

Command Reference for Encounter RTL Compiler

Applets

applet avail

```
applet avail
    [-location applet_installation]
    [-outdated] [-local]
```

Provides information for all the applets available in your local installation and those on the applet server, and includes version information allowing you to see version differences between installed applets and applets on the server.

Options and Arguments

-location *applet_installation*

Specifies the location of the local applet installation from where applets can be loaded.

Specify this option if you want to override the value of the `applet_search_path` root attribute.

By default, the location specified through the `applet_search_path` attribute is used.

-local

Only reports information for applets that exist in the local installation(s).

-outdated

Only reports scripts for which a newer version is available on the server. New applets not present in the local installation are also listed.

Examples

- The following command lists all available applets.

```
rc:/> applet avail
=====
...
=====

Info: Collecting applet server information...
#####
Applets Local/Server information:
  Local Install (/home/me/.localApps/rc)
  Remote Server (<install>/tools.<platform>/lib/applets)

  Name      | Installed | Server Ver. | Summary Description
  -----+-----+-----+-----+
  compare_power |       6 |     1.15 | Generate HTML comparison report of power metrics..
  create_tcse |      5 |     1.42 | generate / load / save testcases
  hermes |      3 |     1.14 | Hermes Design Assistant
  lock |      5 |     1.12 | Guarantee atomic operations using a LOCKFILE...
  manual_assert |      3 |      1.3 | Apply default toggle rate to all synthesis ...
```

NOTE: To install an applet, use 'applet install -location <applets directory> <applet name>'
rc:/>

Command Reference for Encounter RTL Compiler Applets

- The following command lists the outdated applets.

```
rc:/> applet avail -outdated
=====
...
=====

Info: Collecting applet server information...
#####
# Local Install (/home/me/.localApps/rc)
# Remote Server (<install>/tools.<platform>/lib/applets)

Name      | Installed | Server Ver. | Summary Description
-----+-----+-----+
compare_power |       6 |      6.4 | Generate HTML comparison report of power metrics..
lock |     N/A |      1.3 | Guarantee atomic operations using a LOCKFILE, ...

NOTE: To install an applet, use 'applet install -location <applets directory> <applet name>'
rc:/>
```

- The following command lists the local applets.

```
rc:/> applet avail -local
=====
...
=====

Info: Collecting applet server information...
#####
# Local Install (/home/me/.localApps/rc)
# Remote Server (<install>/tools.<platform>/lib/applets)

Name      | Installed | Server Ver. | Summary Description
-----+-----+-----+
compare_power |       6 |      1.13 | Generate HTML comparison report of power metrics..
create_tcase |      5 |      1.42 | generate / load / save testcases
hermes |      3 |      1.14 | Hermes Design Assistant
manual_assert |      3 |      1.3 | Apply default toggle rate to all synthesis ...

NOTE: To install an applet, use 'applet install -location <applets directory> <applet name>'
rc:/>
```

Command Reference for Encounter RTL Compiler

Applets

applet install

```
applet install
    [-location applet_installation]
    [applet_list] [-force]
```

Installs one or more applets and their corresponding dependencies.



An applet is not available for execution until it is loaded.

Options and Arguments

<i>applet_list</i>	Only installs the specified applets. By default, all applets on the server will be installed.
<i>-force</i>	Indicates to overwrite the installed applets. By default, the installed applets are not overwritten.
<i>-location applet_installation</i>	Specifies the location where to install the local applets. Specify this option if you want to override the value of the <i>applet_search_path root</i> attribute. By default, the location specified through the <i>applet_search_path</i> attribute is used.

Example

The following command installs the `compare_gates` applet in the specified location.

```
rc:/> applet install -location ~/rc_ex/applets compare_gates
Info: Collecting applet server information...
Installing applet 'compare_gates'...
Updating applet catalog information...
...
rc:/> applet install -location ~/rc_ex/applets compare_gates
Info: Collecting applet server information...
Installing applet 'compare_gates'...
Error: applet 'compare_gates' is already installed. Please use
      'update' command instead or '-force' switch
```

applet list

```
applet list
```

Displays all applets that have been loaded and that are available for use in the current session

Example

The following example shows the applets that were loaded.

```
rc:/> applet load compare_power
...
rc:/> applet list
=====
...
=====

The following applets have been loaded:
    compare_power
    generate_report
    time_info
```

```
rc:/>
```

Command Reference for Encounter RTL Compiler

Applets

applet load

```
applet load  
  [-location applet_installation]  
  [applet_list]
```

Loads one or more applets and their corresponding dependencies.



An applet is not available for execution until it is loaded.

Options and Arguments

-location applet_installation

Specifies the location of the local applet installation from where applets can be loaded.

Specify this option if you want to override the value of the `applet_search_path root` attribute.

By default, the location specified through the `applet_search_path` attribute is used.

applet_list

Only loads the specified applets.

By default, all applets on the server will be loaded.

Example

The following command loads the `compare_power` applet together with all its dependencies.

```
rc:/> applet load compare_power  
=====  
...  
=====  
  
Sourcing '/install_path/tools.lnx86/lib/applets/compare_power/compare_power.tcl'  
(Tue Aug 23 15:52:34 -0700 2011)...  
Sourcing '/install_path/tools.lnx86/lib/applets/generate_report/  
generate_report.tcl' (Tue Aug 23 15:52:34 -0700 2011)...  
Sourcing '/install_path/tools.lnx86/lib/applets/time_info/time_info.tcl' (Tue  
Aug 23 15:52:34 -0700 2011)...  
Loading applet 'compare_power'...  
rc:/>
```

applet update

```
applet update
  [-location applet_installation]
  applet_list
```

Compares the version and contents of the applets in the local installation with those on the server and updates any applets that have newer versions available on the server. Uninstalled applets need to be installed first.

Options and Arguments

-location applet_installation

Specifies the location of the local applet installation that must be updated.

Specify this option if you want to override the value of the *applet_search_path* root attribute.

By default, the location specified through the *applet_search_path* attribute is used.

applet_list

Only updates the specified applets.

If the specified applet was not yet installed, it will be installed in the specified location.

If you do not specify any applets, all applets on the local installation will be updated without installing any missing ones.

Examples

The following example shows a case where the *time_info* applet is newer on the server and hence updated on the local installation

```
rc:/> applet update
Info: Collecting applet server information...
Info: Updating applet installation /home/myname/.localApps/rc...
Updating applet 'time_info' (1.1 -> 1.2)...
Info: Connecting to server....
Info: Collecting applet server information...
```

applet version

applet version

Provides version information of the applet command

Example

The following command shows the current version of the applet command.

```
rc:>/ applet version  
applet command Version: 4
```

Command Reference for Encounter RTL Compiler

Applets

applet whatis

```
applet whatis [-detail] applet_list
```

Provides a summary description of the capabilities of the specified applets.

Options and Arguments

<i>applet_list</i>	Lists the applets for which you want information.
-detail	Specifies to provide a detailed description of the capabilities of the specified applets.

Examples

- The following command requests summary information for the `time_info` and `lock` applets.

```
rc:/> applet whatis {time_info lock}
=====
disclaimer info...
=====

Info: Collecting applet server information...
Applet: time_info
    Location: install_path/lib/applets
    Version: 2
    Summary: Create time stamp and keep track of runtime metrics

Applet: lock
    Location: install_path/lib/applets
    Version: 2
    Summary: Guarantee atomic operations using a LOCKFILE, wrapper
functions
```

- The following command requests detailed information for the `time_info` applet.

```
rc:/> applet whatis time_info -detail
Info: Collecting applet server information...
Applet: time_info
    Location: install_path/lib/applets
    Version: 2
    Summary: Create time stamp and keep track of runtime metrics

    Full Description:
        This script allows the user to create time stamps such that
        at any time the user can generate a report of the runtime
        allocation throughout a run.
```

Command Reference for Encounter RTL Compiler Applets

Index

A

abstract model
 logic, writing [255](#)
 scan, reading [783](#)
 scan, writing [818](#)
adding
 design [959](#)
 directory to stack [55](#)
 help for user command [982](#)
 user-defined command [990](#)
 user-defined message [987](#)
area, reporting [385](#)
assigning, paths to cost group [318](#)
ATPG file, writing out [583, 754, 805](#)
Attribute
 defining new
 defining custom [983](#)
attributes
 retrieving value [77](#)
 setting value [108](#)

B

bitblasting
 one port [946, 949](#)
bitblasting, all ports [948](#)
body segment, defining [674](#)
boundary scan logic
 previewing changes [719](#)
busses
 renaming [937](#)

C

cells
 reporting cell count [385](#)
 reporting technology library cells
 used [427](#)
changing
 attribute value
 defined by RC [108](#)
 directory in design hierarchy [37](#)
 names

 of instances of an object type [937](#)
path constraints [308](#)
 UNIX working directory [83](#)
channel masking types [578](#)
clock gating
 insertion in mapped netlist [869](#)
clock inputs, reporting [372](#)
clock waveform, defining [290](#)
clock-gating logic
 editing [863](#)
 inserting and connecting observability
 logic [870](#)
 merging instances [866](#)
 removing [874](#)
 reporting [391](#)
 test input, connecting [865](#)
color
 highlighting [131](#)
 label [150](#)
commands, alphabetical list of [19](#)
compatible test clocks, declaring [797](#)
compression logic
 previewing changes [392](#)
concatenating
 scan chains [615](#)
configuring, pads for DFT [617](#)
congestion map
 create snapshot for documentation [153](#)
connecting
 pins, ports [950](#)
 scan chains [620](#)
 test input of clock-gating logic [865](#)
constraints
 checking [500](#)
 reading in [209](#)
cost group
 assigning paths to [318](#)
 defining [295](#)
creating
 binding for Chipware component [170](#)
 Chipware component [172](#)
 delay constraint for specific path [312](#)
 design [959](#)
 design representation in design
 hierarchy [332](#)
 HDL library [176](#)

hierarchy in design [956](#), [968](#)
implementation for Chipware
component [174](#)
instance of [960](#)
library domains (MSV) [915](#)
parameter for Chipware
component [180](#)
pin [182](#)
port bus [962](#)
subdesign [952](#)
support bus [965](#)
synthetic operator [177](#)
timing constraints for instance [296](#)
user-defined message [987](#)
critical path, reporting timing slack for [498](#)
current directory
in design hierarchy [56](#)
in UNIX [93](#)

D

defining
boundary scan segment [633](#)
clock [290](#)
configuration mode [636](#)
cost group [295](#)
DFT test clock [656](#), [693](#)
input and output delays [298](#)
multi-cycle path [304](#)
paths for timing constraints [323](#)
scan clock for LSSD [680](#), [683](#)
shift-enable signal for DFT [686](#)
test-mode signal for DFT [697](#)
user_defined scan chain [674](#)
user-defined abstract segment [628](#)
design
area [385](#)
creating
generic netlist [350](#)
new design [959](#)
incremental optimization [351](#)
instantiating [960](#)
mapping [349](#)
removing [972](#)
renaming [937](#)
synthesizing [348](#)
uniquifying [967](#), [980](#)
design hierarchy
adding hierarchy level [956](#), [968](#)
changing directories in [37](#)

finding object type [44](#)
listing information [50](#)
removing hierarchy level [966](#)
removing objects [946](#), [953](#), [977](#)
renaming objects [972](#)
returning current position [56](#)
design rule constraints
reporting violations on [405](#)
DFT clock domain
associating with scan chain [675](#)
specifying compatible test clocks [797](#)
DFT MBIST clocks
defining [656](#)
DFT rule violations
checking flip-flops for [589](#)
fixing [703](#)
DFT rules, list of rules checked [589](#)
DFT test clocks
declaring as compatible [797](#)
defining [693](#)
directories
changing in
design hierarchy [37](#)
UNIX [83](#)
listing contents in
design hierarchy [50](#)
UNIX directory [90](#)
virtual directory [49](#)
returning current position in
design hierarchy [56](#)
UNIX [93](#)
directory stack
adding new directory [55](#)
displaying directory list [40](#)
tracing previous directory [54](#)
disconnecting pins, ports [954](#)
displaying
current position in design hierarchy [56](#)
directory stack [40](#)
UNIX directory, contents [90](#)
virtual directory, contents [49](#)

E

executing
script [82](#)
UNIX shell command [110](#)
exiting, from RTL Compiler [76](#)

F

fanin, reporting [374](#)
fanout, reporting [377](#)
filtering, objects [41](#)
find, object type [44](#)
finding
 corresponding input or output [48](#)
 object base name [36](#)
 object directory name [39](#)
 object type [44](#)
 type of object [60](#)
fixing, DFT rule violations [703](#)
flattening, instances [978](#)
flip-flops
 checking for DFT rule violations [589](#)
 reporting scannability [415](#)
Floorplan (DEF)
 Reading [545](#)
 Writing [566](#)

G

generic netlist
 creating [350](#)
GUI
 update [125](#)

H

HDL files, reading [201](#)
HDL Viewer
 remove data [127](#)
head segment, defining [676](#)
help
 adding for new command [982](#)
 providing for commands [65, 81](#)
highlight
 clear [130, 141](#)
 color [131, 148](#)

I

incremental optimization [351](#)
input delay, defining [298](#)
inserting
 clock-gating logic in mapped netlist [869](#)

shadow logic [763](#)
test point [768](#)
user-defined test point [773](#)
wrapper cell [775](#)
instances
 grouping [956, 968](#)
 removing from design [972](#)
 renaming [937](#)
 ungrouping [966](#)
isolation logic
 removing [916](#)
isolation rule
 removing [972](#)

L

leakage power, reporting [462](#)
length
 abstract segment [629](#)
 maximum chain length [677](#)
level-shifter groups
 removing [972](#)
library cell, instantiating [960](#)
library domains
 creating [915](#)
licenses
 checked out, listing [88](#)
 checking out [86](#)
listing
 object information [50](#)
lockup element type, chain-specific [678](#)
log file [100](#)
log file, specifying [100](#)
logic abstract model, definition [255](#)
logic abstract, writing [255](#)
loop breakers
 removing [974](#)
 reporting [389](#)

M

mapped netlist
 creating [351](#)
 writing out [255](#)
mapping
 design [348](#)
 non-scan flops with scan-equivalent [787](#)
memory resources, reporting [442](#)

messages

- creating customized [987](#)
- reporting [444](#)
- sending [989](#)
- suppressing [113](#)

mode, specifying for power, timing analysis, and optimization [287](#)

MSV design

- library domains, creating [915](#)

multi-cycle path, defining [304](#)

N

names

- adding prefix [939](#)
- limiting length [939](#)
- mapping characters [939](#)

net power, reporting [462](#)

nets

- renaming [937](#)

O

object

- finding base name [36](#)
- finding directory name [39](#)

object path

- return a selection list for instance, net, pin, and port objects [123](#)

object type, finding [44](#)

object types

- listing all attributes for a type [77](#), [108](#)

objects

- connecting [950](#)
- disconnecting [954](#)
- filtering on attribute value [41](#)
- instantiating [960](#)
- listing information for an object [50](#)
- removing from design hierarchy [946](#), [953](#), [977](#)
- renaming [972](#)

returning type of an object [60](#)

selecting in design hierarchy [41](#)

observability logic for clock-gating logic inserting [870](#)

OPCG logic

- previewing changes [749](#)

OPCG segments

- previewing connections [619](#)

operand-isolation logic

- reporting [456](#)

optimizing

- for area or timing [341](#)
- incrementally [348](#)
- RTL [349](#)

output delay, defining [298](#)

output, redirecting [103](#)

P

parameterized module, elaborating [332](#)

path

- return path for a selected object [123](#)

path constraints, modifying [308](#)

paths

- assigning to cost group [318](#)
- creating for timing exception [323](#)
- unconstraining [315](#)

pins

- connecting manually [950](#)
- disconnecting [954](#)

port buses

- creating new object [962](#)
- removing from design [972](#)
- renaming [937](#)

ports

- bitblasting [948](#)
- connecting manually [951](#)
- disconnecting [954](#)
- renaming [937](#)

power

- reporting [462](#)
- sorting report [464](#)

power domain

- removing [972](#)

primitive cell, creating [963](#)

probability values of nets

- reading or updating [885](#)
- writing to SAIF file [901](#)
- writing to TCF file [903](#)

R

reading

- HDL files [201](#)
- SAIF file [880](#)
- SDC constraints [209](#)
- TCF file [885](#)

Command Reference for Encounter RTL Compiler

remove
 data [127](#)

removing
 clock-gating logic [874](#)
 directory from stack [54](#)
 hierarchy from design [966](#)
 object from design hierarchy [953, 977](#)

renaming
 instances of object type [937](#)
 object [972](#)

reporting
 area of design [385](#)
 cell types, used [427](#)
 clock information of design [397](#)
 clock-gating information [391](#)
 design rule violations [405](#)
 DFT registers [415](#)
 DFT setup information [419](#)
 DFT violations [423](#)
 inferred datapath operators [400](#)
 instance information [432](#)
 memory resources [442](#)
 messages in current session [444](#)
 net information [452](#)
 operand-isolation information [456](#)
 port information [460](#)
 scan chains [406](#)
 timing information [498](#)

retrieving
 attribute value
 defined by RC [77](#)
 clock ports [372](#)
 fanin information [374](#)
 fanout information [377](#)
 input ports [67](#)
 object information [50](#)
 output ports [68](#)
 UNIX working directory [93](#)

RTL Compiler
 exiting [76](#)
 quit [97](#)
 resuming process [114](#)
 starting from UNIX [98](#)
 suspending process [114](#)

RTL optimization [349](#)

RTL power analysis, enabling [464](#)

reading [880](#)
writing [899](#)

scan abstract model, creating [818](#)
scan abstract model, reading [783](#)

scan chain
 tool-created
 test clock domain associated with name [641, 643, 671, 689, 710](#)

scan-chain configuration
 previewing [623](#)
 reporting [406](#)

scan chains
 connecting [620](#)
 defining [674](#)
 mixing edges of same clock on [797](#)
 removing [972](#)
 reporting [406](#)

scan data input
 creating [675](#)
 specifying for chain [677](#)
 specifying for scan segment [630, 672](#)

scan data output
 creating [675](#)
 specifying for chain [677](#)
 specifying for scan segment [630, 672](#)
 using existing port [612, 678, 699, 729](#)

scan segment
 defining [628, 641, 643, 671, 689, 710](#)
 removing [972](#)

shift enable
 confirming if connected [629, 672](#)

using in scan chain
 as body segment [674](#)
 as head segment [676](#)
 as tail segment [678](#)

scanDEF file, writing out [856](#)

scripts
 executing [82](#)
 writing out [262](#)

SDC constraints
 reading in [209](#)
 unsupported constructs [209](#)
 writing out [265](#)

shadow logic
 inserting [763](#)
 previewing changes [765](#)

shift-enable port
 for scan segment [630](#)
 confirming if connected [629, 672](#)
 for shift-enable signal [667, 687](#)

shift-enable signal

S

SAIF file

Command Reference for Encounter RTL Compiler

defining [686](#), [697](#)
specifying default [686](#)
specifying for chain [678](#)
slack, reporting at timing endpoints [499](#)
SPEF
 Reading [551](#)
 Writing [568](#)
subdesigns
 creating by copying existing
 subdesign [952](#)
 instantiating [960](#)
 renaming [937](#)
 uniquifying [967](#), [980](#)
subport buses
 creating new object [965](#)
 removing from design [972](#)
 renaming [937](#)
subports
 connecting manually [951](#)
 disconnecting [954](#)
switching power, reporting [462](#)
synthesis
 creating generic netlist [350](#)
 incremental optimization [351](#)
 mapping [351](#)

T

tail segment, defining [678](#)
TCF file
 reading [885](#)
 writing [903](#)
test mode signal, defining [697](#)
test point
 inserting [768](#)
 possible types [770](#)
timing constraints
 deriving for instance [296](#)
 reporting violations on [498](#)
 writing out [262](#)
timing exception, created by
 modifying path constraints [308](#)
 overriding default clock edge
 relationship [304](#)
 specifying timing constraints [312](#)
 unconstraining paths [315](#)
timing units [498](#)
toggle counts of nets
 reading or updating [885](#)
 writing to SAIF file [901](#)

writing to TCF file [903](#)

U

uniquifying, design or subdesign [967](#), [980](#)
UNIX shell command
 executing in RTL Compiler [110](#)
user-defined test point, inserting [773](#)
utilization map
 create snapshot for documentation [153](#)

W

wireload model, reporting [385](#)
worst paths, reporting timing for [501](#)
wrapper cell, inserting [775](#)
writing
 compression model [811](#)
 DFT abstract [818](#)
 logic abstract [255](#)
 netlist file [255](#)
 SAIF file [899](#), [901](#)
 scanDEF file [856](#)
 SDC constraints [265](#)
 TCF file [903](#)
 timing and design rule constraints [262](#)